SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Hrvoje Macura

Zagreb, 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Bojan Jerbić, dipl. ing.

Student:

Hrvoje Macura

Zagreb, 2015.

IZJAVA

Izjavljujem da sam ovaj rad izradio samostalno na Fakultetu strojarstva i brodogradnje u Zagrebu koristeći stečena znanja tijekom studija i navedenu literaturu.

Hrvoje Macura

ZAHVALA

Zahvaljujem se svome mentoru prof.dr.sc. Bojanu Jerbiću i asistentu Filipu Šuligoju mag.ing. koji su iskazali povjerenje te svojim znanstvenim i stručnim savjetima oblikovali temeljnu ideju ovog rada i pomogli mi u njegovoj realizaciji.

Želim se također zahvaliti i svim ostalim djelatnicima i asistentima na Katedri za projektiranje izradbenih i montažnih sustava na suradnji, ugodnom boravku i svim korisnim sugestijama pruženim prilikom izrade diplomskog rada.

Na kraju bih se još posebno zahvalio svojoj obitelji i svim kolegama na iskazanom povjerenju, strpljenju te moralnoj podršci koju su mi ukazali tokom studija.

H. M.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **Hrvoje Macura** Mat. br.: 0035177478

Naslov rada na hrvatskom jeziku: **Automatsko upravljanje sustava robotske glave i ruke**

Naslov rada na engleskom jeziku: **Automatic control of a robot head and arm**

Opis zadatka:

Potreba za intuitivnom interakcijijom sa strojevima nalaže razvoj "humanoidnih" elemenata tehničkog sustava koji pomažu prijenosu informacija. U okviru diplomskog rada potrebno je izraditi, i programirati humanoidni sustav robotske ruke UR5 i robotske glave s dva stupnja slobode, upravljane Arduino mikrokontrolerom, kućišta, elemenata prijenosa i Microsoft Kinect kamere. Razvijeni sustav treba osigurati sljedeće funkcionalnosti:

- automatska kalibracija koračnih motora robotske glave,
- mogućnost komunikacije između robotske upravljačke jedinice, Microsoft Kinecta i Arduino mikrokontrolera,
- automatsko pomicanje robotske glave i centriranje Microsoft Kinect kamere u smjeru vrha robotske ruke,
- automatsko pomicanje glave robota u smjeru glave čovjeka koji se nalazi u vidnom polju.

Zadatak zadan: 7. svibnja 2015.

Rok predaje rada: 9. srpnja 2015.

Predviđeni datum obrane: 15., 16. i 17. srpnja 2015.

Zadatak zadao:

Prof. dr. sc. Bojan Jerbić

Predsjednik Povjerenstva:

Prof. dr. sc. Franjo Cajner

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

Razvoj tehnika upravljanja robotskih sustava započeo je paralelno sa razvojem tehničke grane robotike. Do danas su se početni koncepti upravljanja takvim sustavima značajno izmjenili te i dalje teže sve većem stupnju autonomnosti i inteligencije.

U ovom radu, temeljem iste ideje, izrađena je dvoosna robotska glava koja je povezana u sustav sa industrijskom robotskom rukom te je programski ostvareno automatsko upravljanje sustava. Za potrebe upravljanja je razvijena programska podrška u C# programskom jeziku sa primjenom Kinect senzora, koji se nalazi u sklopu robotske glave, te funkcijom praćenja korisnika u prostoru iz koje se dobivaju potrebne informacije za ostvarivanje upravljanja. Aplikacijom je omogućena i komunikacija u lancu: robotska ruka – aplikacija – robotska glava.

U radu su prikazana rješenja za izvršavanje dva zadatka ovog sustava, praćenje korisnika „pogledom“ prilikom kooperacije korisnika sa sustavom robotske glave i praćenje izvršavanja raznih zadataka u kooperaciji sa robotskom rukom.

Širok spektar mogućnosti ovog sustava primjenom navedenih elemenata predstavlja odličnu platformu za ostvarivanje sličnih zadataka uz mogućnost proširenja i daljnjeg unapređivanja sustava dodavanjem novih elemenata.

Ključne riječi: robotika, automatsko upravljanje, robotska glava, robotska ruka, C#, Kinect, aplikacija

SUMMARY

Development of robotic system control methods has started simultaneously with beginning of robotic system development. Initial concepts for controlling such systems were significantly changed until now and they aim to have even higher degree of autonomy and intelligence.

In this paper, based on the same idea, robotic head with two axis is made and it's connected with industrial robotic arm in one system which is automatically controled by developed program. This program is developed in C# programing language and it's used for control purposes using a Microsoft Kinect sensor which is integrated in robotic head assembly. The program uses Kinect's skeletal tracking function to obtain values required for control realization. In this main program is also enabled communication in control chain: industrial robotic arm – main aplication – robotic head.

In this paper are also shown solutions for two controlling tasks with this system. First task is cooperation of robotic head with user, where the main idea is to track user and try to direct robotic head's „face“ according to user's head position. Second task is, in cooperation of robotic head with robotic arm, to direct robotic head's „face“ according to robot's tool center point (TCP) position.

A wide range of possibilities of this system with using all mentioned elements represent excellent platform for realization more similar tasks and possibility for future system upgrade and further improvement.

Key words: robotics, automatic control, robotic head, industrial robotic arm, C#, Kinect, application

1. UVOD

Na početku razvoja tehničke grane robotike svi su se uređaji, koji su obavljali automatizirano neku zadaću, nazivali robotima. Kasnije se, daljnjim razvojem kroz vrijeme, definicija robota promijenila kao i cijela podjela grane robotike. U posljednje vrijeme se robotika uglavnom dijeli na nekoliko grana od kojih su najznačajnije grane industrijske robotike, mobilne robotike i uslužne robotike ali se sve više ove grane međusobno isprepliću te je gotovo nemoguće napraviti konačnu podjelu robotike. Industrijska robotika obuhvaća uglavnom razne izvedbe robotskih ruku sa 2-3 i više stupnjeva slobode gibanja i primjenjuju se najčešće za razne potrebe u industrijskim primjenama. Uglavnom su zadaci manipulacija i obavljanje određenih operacija na proizvodima, obavljanje radnih zadataka na nekih mjestima koja su opasne prirode za čovjeka (radijacija, visoke temperature, opasne kemikalije i sl.) ili za obavljanje monotonih zadaća uz osiguranje jednolične kvalitete obavljanja zadatka.

Mobilna se robotika uglavnom zasniva na automatiziranim uređajima koji objedinjavaju razne elemente i senzore te nisu fiksno ograničeni za mjesto obavljanja radnje već imaju određeni radni prostor unutar kojega se mogu gibati te izvršavati zadatke manje kompleksnosti. U granu uslužne robotike uvrštavaju se razne izvedbe uređaja, često povezanih sa područjem mobilne robotike, a u osnovi zadaća im je uslužna djelatnost tj. obavljanje raznih korisnih zadataka za čovjeka. Tako se u ovoj skupini nalaze sve popularniji autonomni robotski usisavači, terapeutski roboti, roboti za medicinske potrebe i sl.

Iako je uslužna robotika navedena posljednja, sklonost ljudi prema robotima je kod nje daleko veća nego što je to u slučaju primjene industrijskih robota. Upravo se zbog toga u primjenama industrijskih robota u kooperaciji s korisnicima želi izbjeći ta odbojnost primjenjujući razne „humanoidne“ elemente dajući tako industrijskom robotu određenu razinu osobnosti. Prema tome se i temelji cijeli daljnji razvoj grane robotike s ciljem postizanja većeg stupnja autonomije robota i razvijajući inteligentne robotske sustave.

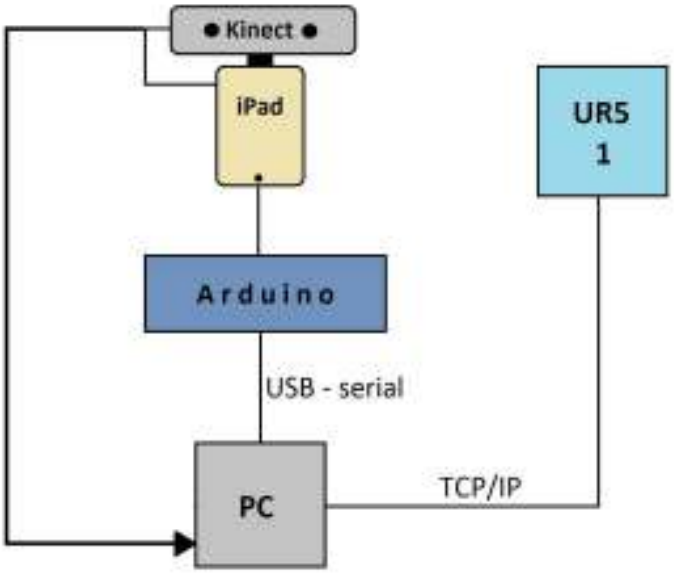
Upravo je iz slične ideje nastao i ovaj rad. Kako bi u primjenu industrijske robotske ruke unijeli malo osobnosti za pristupačniji rad sa korisnikom odlučeno je izraditi robotsku glavu i njeno gibanje upariti sa radom korisnika i robotske ruke. Glava je zamišljena kao mehanizam sa dva stupnja slobode gibanja, na sebi bi sadržavala jednostavnu zamjenu za lice u obliku tableta sa prikazom različitih ekspresija lica a kao sustav za navođenje i promatranje okoline,

kao oči bi se koristio Kinect senzor. Pogon glave bi činio po jedan koračni motor za svaku od dviju osi. Za upravljanje je izabrana Arduino mikrokontrolerska pločica u kombinaciji sa odgovarajućim upravljanjem za izabrane koračne motore.

Jedan dio rada obuhvaća korištenje Kinect senzora i njegove funkcije za prepoznavanje korisnika koji se nalazi u radnom prostoru te prećenje njegovog gibanja po prostoru uz stalno usmjeravanje pogleda u smjeru korisnikove glave.

Drugi dio rada predstavlja kombinacija paralelnog rada industrijske robotske ruke i gibanja robotske glave u smjeru vrha alata (TCP-a) kako bi se što vjernije imitiralo ljudsko ponašanje usmjeravanjem pogleda prema mjestu izvršavanja radnje.

Ovo su dva temeljna zadatka koja će biti opisana u ovom radu a uz sve mogućnosti koje sustav posjeduje sa pripadajućim elementima predstavlja odličnu platformu za razne daljnje aplikacije i zadatke koje je u mogućnosti izvršavati. Upravljački sustav robotske glave je također izabran s obzirom na fleksibilnost koju omogućava te je veoma jednostavno cjelokupni sustav proširiti sa dodatnim senzorima ili pokojim aktuatorom za proširenje mogućnosti robotske glave.



Slika 1. Korišteni elementi u sustavu automatskog upravljanja robotskom glavom

Slika iznad prikazuje shematski elemente sustava sa vezama između njih, no kako je već spomenuto prije, omogućeno je proširenje sustava dodatnim elementima te razvoj drugih aplikacija u širokom spektru mogućnosti.

2. SUSTAV AUTOMATSKOG UPRAVLJANJA ROBOTSKOM RUKOM I GLAVOM

Sustav automatskog upravljanja robotskom rukom i robotskom glavom uz korištenje Kinect-a za obradu slike i ostvarivanje funkcije automatskog gibanja robotske glave u ovisnosti o preostalim elementima. Sustav se sastoji od robotske glave kao glavnog dijela sustava te komercijalne industrijske robotske ruke koja radi u kooperaciji s robotskom glavom. Sklop robotske glave je izveden kao dvoosni mehanizam pogonjen s dva koračna motora. Na robotskoj glavi se nalazi Kinect za vizualno primanje informacija iz okoline te ispod njega tablet koji slikovno prikazuje ekspresije lica, dajuću „humanodni“ karakter cijelom sustavu. U sustavu se nalazi i prikladni upravljački dio koji upravlja gibanjem mehanizma robotske glave. Sve nevedene elemente sustava povezuje nadređeno računalo na kojem se izvršava aplikacijski dio sa Kinect-om i komunikacija sa robotskom rukom i upravljačkim dijelom robotske glave.

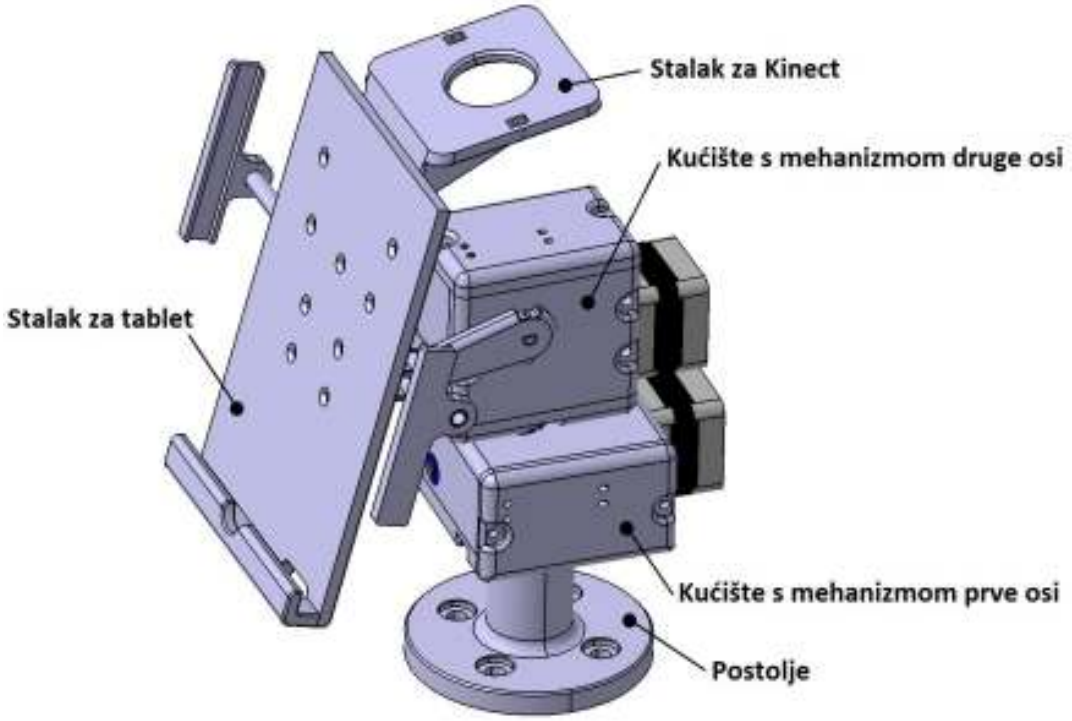
Za sustav je zamišljeno da izvršava zadatke kooperacije rada sa korisnikom i sa robotskom rukom. U radu s korisnikom potrebno je da robotska glava pomoću Kinect-a smještenog na njoj, prepozna glavu korisnika u vidnom polju, te prema njenom položaju u prostoru automatski upravlja gibanjem robotske glave usmjeravajući „lice“ robotske glave prema korisniku. Slično tome je potrebno ostvariti i kooperaciju s robotskom rukom, u kojoj će robotska ruka obavljati neku zadaću a robotska će glava pri tome pratiti rad robotske ruke usmjeravajući „pogled“ prema TCP-u robotske ruke, „nadzirući“ tako obavljanje zadaće.

3. SUSTAV ROBOTSKE GLAVE

Robotska glava je izvedena kao robotizirani mehanizam sa dva rotacijska stupnja slobode gibanja aktuirana sa dva koračna motora. Mehanizam je predviđen za korištenje i ostvarivanje funkcija zakretanja glave robota, a rad i upravljanje radom glave odvija se u kombinaciji sa drugim elementima kako bi se što bolje prikazale i oponašale karakteristike ljudske glave odnosno njenog gibanja i funkcija. Ostvarivanjem ovih funkcija za rad robotskog sustava njegovo se djelovanje približava korisniku te se robot s takvim funkcijama bolje implementira u okruženje gdje se odvija interakcija između ljudi i uobičajenih robotskih sustava. Ovim mehanizmom bi se oponašale funkcije ljudske glave na način da je na mehanizmu postavljen tablet na kojem bi se prikazivale ekspresije lica a funkcije vida i sluha bi se iskoristile primjenom Kinect senzora o kojem će više riječi biti u jednom od sljedećih poglavlja.

3.1. Dijelovi sustava robotske glave

Robotska glava se sastoji od nekoliko dijelova, ugrubo podijeljeni to su: postolje, kućište sa mehanizmom prve osi gibanja, kućište sa mehanizmom druge osi gibanja, stalak za iPad i stalak za Kinect senzor.



Slika 2. CAD sklop robotizirane glave

Postolje mehanizma ima 4 provrta i služi za montažu na neku podlogu kako bi mehanizam bio stabilan za rad i bez nepoželjnih vibracija. Na postolje se nastavlja dio kućišta sa mehanizmom za ostvarivanje prvog stupnja slobode gibanja odnosno zakretanje oko prve osi i to u horizontalnoj ravnini. S vanjske strane kućišta postavljen je pogonski motor, čije se gibanje prenosi unutar kućišta, preko prijenosnog mehanizma na iduće dijelove sustava. Na malo vratilo koje izlazi iz kućišta prvog stupnja slobode gibanja se nastavlja kućište sa mehanizmom za ostvarivanje drugog stupnja slobode gibanja. Na ovom se kućištu također s vanjske strane nalazi smješten pogonski motor čije se gibanje unutar kućišta, prijenosnim mehanizmom, prenosi vratilom ostvarujući tako drugi stupanj slobode gibanja zakretanjem oko druge osi u vertikalnoj ravnini. Na to vratilo koje izlazi s dvije strane iz kućišta drugog stupnja slobode gibanja se nastavljaju, sa svake strane po jedna polugica koje drže stalak za tablet i stalak za Kinect kao zadnji dio mehanizma robotizirane glave.

Nakon početnih proračuna [1][1] momenta potrebnog za prijenos i dimenzioniranje prijenosnog mehanizma je uslijedio korak konstruiranja ostalih nestandardnih elemenata kućišta i mehanizma te modeliranje u programskom CAD paketu Catia-i. Konačni sklop sa svim dijelovima robotiziranog mehanizma robotske glave prikazan je na prethodnoj slici (**Slika 2**). Konačna konstrukcija je prije postupka izrade malo dorađena usmjeravajući pažnju na tehnologičnost pri oblikovanju elemenata kako bi se isti kasnije mogli što jednostavnije proizvesti. Za proizvodnu tehnologiju je odlučeno koristiti postupak 3D printanja koji je u proteklih desetak godina znatno napredovao te nam može osigurati potrebnu točnost izrade dijelova koja je uvjetovana primjenom ležaja i pojedinih elemenata prilično malih dimenzija.

Nakon izrade svih dijelova sustava, izvršena je montaža dijelova u podsklopove i konačni sklop robotske glave.

Slika 3. Sklop mehanizma robotske glave

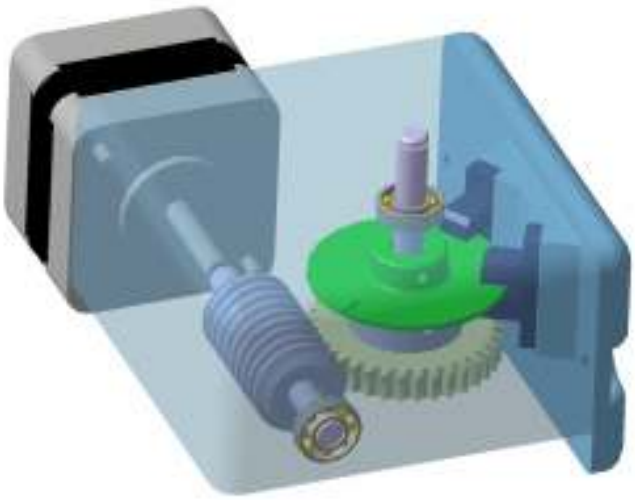
Zbog pojednostavljenja sustava i jednostavnije nabavke svih potrebnih dijelova odlučeno je umjesto kupnje gotovih elemenata prijenosnog mehanizma iste izraditi također postupkom 3D printanja. Također su sva vratila mehanizma umjesto od metala izrađena od dovoljno čvrstog polimera.

Prvenstveno zamišljeni stalak za tablet je predviđen kao fleksibilno rješenje sa mogućnošću prihvata tableta raznih dimenzija kako nebi bili ograničeni samo na primjenu sa jednim modelom tableta. Kako je na raspolaganju bio i jedan držač za raspoloživi tablet iPad iznimno jednostavne i lagane konstrukcije odlučeno je za početak, umjesto izrade prvotno predviđenog stalka, njega iskoristiti za ovu funkciju. Ovaj posebni držač, crne boje, je vidljiv na slici sklopa robotske glave (**Slika 3**).

3.2. Prijenosni mehanizam

Za pogon i zakretanje svake od dviju osi potreban je određeni moment. Proračun i razrada prijenosnog mehanizma provedeni su u okviru drugog rada te se ovdje neće ponovno izvoditi već će se ovdje samo navesti osnovne karakteristike prijenosnog mehanizma za ovaj slučaj.

U obzir su bile uzete sve vrste prikladnih prijenosa. Sa poznatim momentom motora i za postizanje potrebnog momenta za zakretanje osi remenski je prijenos zahtijevao velike dimenzije remenica te je iz tog razloga za prijenosni mehanizam izabran pužni prijenos. On je uz zadovoljavajući iznos momenta imao i manje dimenzije tako da je cijeli prijenosni mehanizam zbog toga vrlo kompaktan.



Slika 4. Izgled prijenosnog mehanizma

Zbog konstrukcije i položaja osi kod prvog stupnja slobode gibanja, iako on preuzima gotovo cijelu težinu robotske glave, većina opterećenja djeluje u aksijalnom smjeru izlaznog vratila te to aksijalno opterećenje preuzima ležaj na donjem kraju vratila. Radijalnu komponentu opterećenja preuzima motor preko pužnog prijenosa. Kod prvog stupnja slobode gibanja je u stacionarnom stanju sustav prijenosa neopterećen jer prilikom mirovanja nema radijalnog opterećenja a aksijalno opterećenje preuzima ležaj.

Kod drugog stupnja slobode gibanja je izlazno vratilo uležišteno sa dva radijalna ležaja a aksijalni pomak je spriječen čvrstim spojem polugica sa svake strane vratila. Ovo je vratilo konstantno opterećeno zbog djelovanja gravitacijske sile svih elemenata koji se vežu preko polugica na drugu os gibanja. Kod ove je osi radijalno opterećenje preko pužnog kola na pužni vijak prilično veliko dok je aksijalno opterećenje gotovo zanemarivo. Trenje na dodiru pužnog vijka i pužnog kola treba biti čim manje kako bi se što manje momenta od motora gubilo na savladavanje tog trenja i time uzrokovanog povećanog trošenja elemenata.

Ovi elementi izrađeni postupkom 3D printanja nemaju najpoželjniju kvalitetu površine što uzrokuje nešto veće trenje.

Nedovoljna brzina prijenosa – izbor novog omjera [2]

Testiranjem motora koji su bili na raspolaganju pokazalo se da motori imaju dosta manji iznos maksimalnog broja okretaja od očekivanog te on iznosi 2,5 s^{-1}. Omjer pužnog prijenosa (i) se računa kao odnos broja zuba pužnog kola i broja zuba pužnog vijka i ovdje iznosi 30:1. Poznavajući omjer pužnog prijenosa i broja okretaja pužnog vijka koji je povezan na vratilo motora, jednostavnim proračunom dobijemo broj okretaja pužnog kola:

$$n_{PK} = \frac{n_{PV}}{i} = \frac{2{,}5\,\frac{o}{s}}{30} = 0{,}083\,\frac{o}{s} \qquad \textbf{(1)}$$

Iz ovoga se vidi da se pužno kolo zakreće brzinom od 30°/s te za prolaz cijelog radnog područja prve osi u iznosu od 180° zahtijeva 6 sekundi. Ova brzina zakretanja osi je premala da bi imalo vjerno slijedila tj. oponašala zakretanje glave. Iz tog razloga odlučeno je što jednostavnije to izmijeniti te promijeniti omjer pužnog prijenosa. Kako bi dobili iznos brzine zakretanja osi od barem 90°/s odnosno 0,25 s^{-1} i iz poznatog broja okretaja motora odnosno pužnog vijka možemo izračunati potrebni omjer:

$$i = \frac{n_{PV}}{n_{PK}} = \frac{2{,}5\,\frac{o}{s}}{0{,}25\,\frac{o}{s}} = 10 \qquad \textbf{(2)}$$

Zbog razmaka između osi i raspoloživog prostora koji je određen pri konstruiranju kućišta za postizanje približno ovog omjera odlučeno je koristiti pužno kolo sa 22 zuba i pužni vijak sa dva zuba. Time je postignut omjer prijenosa 11:1 čime je zadovoljena brzina zakretanja uz poštivanje dimenzija za mogućnost ugradnje u postojeće kućište.

3.3. Pogonski dio sustava [3]

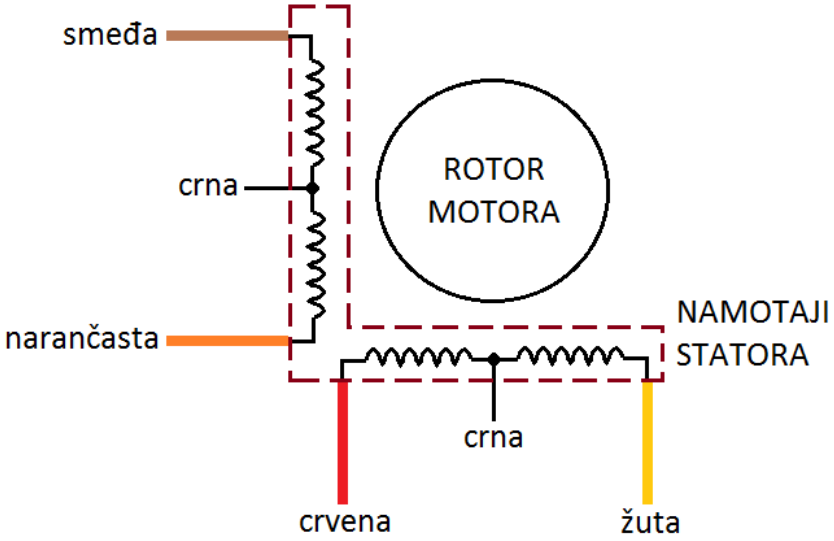
Glava humanoidnog robota je konstruirana kao mehanizam sa dva stupnja slobode gibanja. Svaki stupanj slobode pogoni se jednim aktuatorom. Za aktuatore su izabrani raspoloživi unipolarni 4-fazni koračni motori proizvođača Minebea, model motora je 17PM-M012-13.

Detaljnije karakteristike dane su u sljedećoj tablici.

Tablica 1. Tehničke karakteristike koračnog motora [3]

Minebea Hybrid 17PM-M012-13	
Napon napajanja	6 – 24 V
Struja motora (pri 6 V)	0,67 A
Moment držanja	0,08336 Nm
Fizički kut koraka motora	1,8°
Točnost koraka	+/- 5%
Masa	250 g
Dimenzije osovine	Ø5mm x 22mm

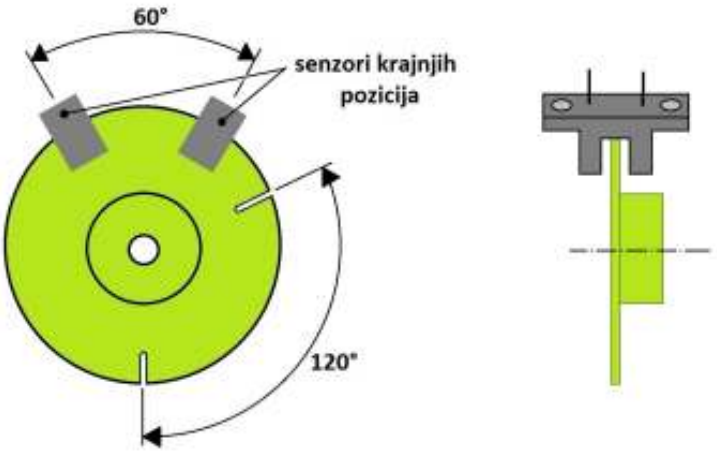
Motor se spaja u električni krug sa 6 žica koje izlaze iz kućišta motora. Iz dokumentacije proizvođača motora i iz provedenog mjerenja može se zaključiti da 4 žice pripadaju namotima motora a ostale 2 žice su na istom potencijalu te predstavljaju spoj na masu. Proizvođač motora najčešće daje samo simbolički prikaz izvedbe namotaja i priključaka motora, a ukoliko se želi napraviti drugačije upravljanje motorom u koračnom ili polukoračnom režimu korisna je informacija i poznavanje fizičkog rasporeda faza u motoru.



Slika 5. Shema rasporeda faza motora

3.4. Sustav za inicijalizaciju robotske glave

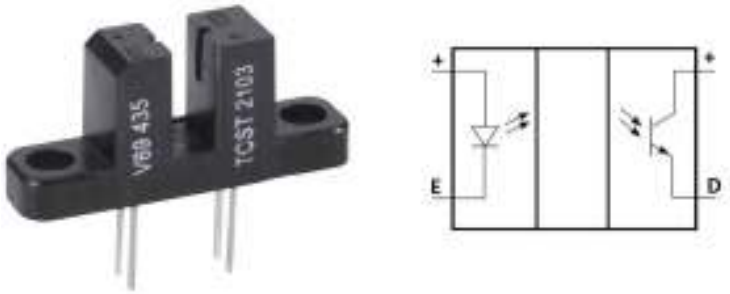
Sustav mehanizma je pogonjen koračnim motorima koji se upravljaju u koračnom režimu tj, zakret rotora se odvija korak po korak. Poznavajući fizički izvedbu konstrukcije motora, svaki korak iznosi 1,8° a za zakret od 360° to iznosi 200 koraka motora. Iako se koraci mogu programski brojati i s njima određivati relativnu poziciju sustava tj. pomak u odnosu na neku poznatu poziciju. Apsolutno određivanje položaja sustava je praktično nemoguće izvesti ako ne znamo početnu poziciju sustava u odnosu na koju se onda mjeri svaki daljnji pomak. Iz tog se razloga u ovaj mehanizam integrirao sustav za inicijalizaciju tj. za postavljanje sustava u određenu početnu točku.



Slika 6. Sustav diska i senzora za inicijalizaciju

Sustav se sastoji (Slika 4) od diska postavljenog na os koja predstavlja jedan stupanj slobode gibanja i 2 optička senzora za indikaciju krajnjih pozicija zakreta osi. Kako je spomenuto u jednom od prethodnih poglavlja radno područje zakreta vertikalne osi iznosi ± 90°. Na disku se nalaze dva utora zamaknuta za 120° a svaki od njih je za aktivaciju jedne krajnje pozicije. Senzori su također zamaknuti i to za 60° što u kombinaciji sa zamakom utora na disku ostavlja radno područje iznosa 180° odnosno ± 90°. Kod druge, horizontalne osi, je radno područje dosta manje zbog veličine sklopa na kraju: stalka za tablet i nosača Kinect-a. Kod ove osi radno područje ukupno iznosi 80° od jedne do druge krajnje pozicije zakreta te je prema tome je i disk izveden tako da su utori zamaknuti za 100° više nego kod prve osi tj. 220° ukupno.

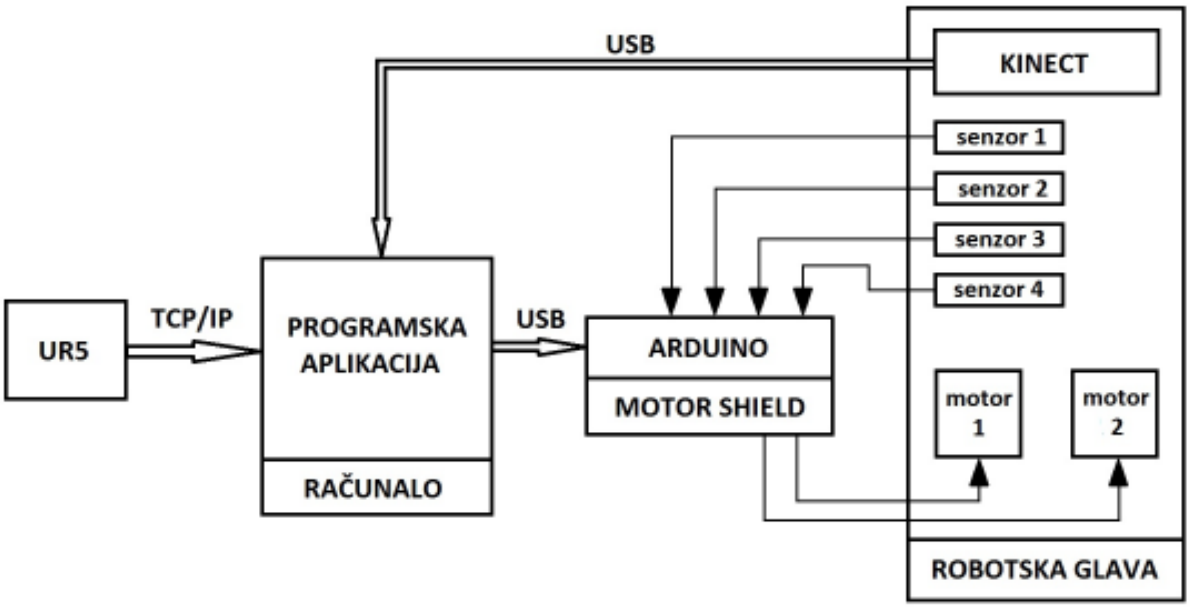
Kao senzor krajnje pozicije korišten je optosprežnik [4], oznake TCST2103, sa konstrukcijski razdvojenim dijelovima svjetleće diode i fototranzistora s druge strane. Prostor između je ostavljen za postavljanje barijere kojom se signalizira prohodnost optičkog puta od jedne do druge strane.



Slika 7. Optički senzor TCST2103 [4]

4. UPRAVLJAČKI DIO ROBOTSKE GLAVE

U ovom radu potrebno je bilo odabrati nekakvu upravljačku jedinicu koja će upravljati radom cijelog sustava, točnije upravljati pogonom motora postolja i koja će komunicirati sa nadređenim računalom. Za izbor upravljačke jedinice postavljeni su uvijeti da posjeduje dovoljan broj digitalnih ulaza odnosno izlaza, da omogućuje komunikaciju sa računalom te naravno da omogućava spajanje i upravljanje koračnim motorima. Kao centralni dio ovog sustava tj. upravljačku jedinicu izabrana je raspoloživa mikrokontrolerska razvojna pločica Arduino a model pločice je Uno.



Slika 8. Upravljački lanac cijelog sustava

4.1. Arduino – upravljačka jedinica [5]

Mikrokontrolerska razvojna pločica Arduino predstavlja moderno rješenje za ostvarivanje raznih zadataka širokog spektra tehničkog djelovanja. Karakteristike su relativno jednostavan i brz razvoj tehničkih sustava, a moguće kompleksnosti sustava variraju od malih sustava poput mjerenja temperature, pogona malih motora i sl. pa sve do većih sustava upravljanja većim elementima, naponima i strujama i to indirektno pomoću posrednih elemenata (releja, tranzistora, optocouplera i sl.). Ovaj model pločice zadovoljava sve navedene uvjete, pločica je relativno mala i kompaktna i predstavlja fleksibilan razvojni sustav koji se jednostavno nadograđuje i modificira po potrebi.

Slika 9. Arduino Uno mikrokontrolerska pločica [5]

Najvažnije karakteristike pločice prikazane su u tablici u nastavku (**Tablica 2**).

Tablica 2. Karakteristike Arduino Uno mikrokontrolerske pločice

Mikrokontrolerska razvojna pločica Arduino Uno	
Mikrokontroler	Atmega328
Radni napon sustava	5 V
Napon napajanja pločice	7 – 12 V
Broj digitalnih ulaza / izlaza (broj raspoloživih PWM kanala)	14 (6)
Broj analognih ulaza	6
Maksimalna struja ulaza / izlaza	60 mA
Iznos flash memorije	32 KB
Iznos SRAM memorije	2 KB
Iznos EEPROM memorije	1 KB
Radna frekvencija sustava	16 MHz

Za potrebu osiguranja komunikacije sa računalom, ova pločica posjeduje USB priključak B tipa, preko kojega se pločica spaja na računalo te se ovim putem vrši uz komunikaciju i programiranje ATmega mikrokontrolera na pločici. Na pločici još postoje dva priključka sa oznakama Rx i Tx (od eng. Receive – primiti i eng. Transmit - poslati) za ostvarivanje serijske veze sa nekim drugim uređajem ili elementom. Od ostalih vrsta komunikacije koju omogućava pločica tu su još priključci za I^2C i SPI komunikaciju.

4.1.1. Programiranje

Rad i sve raspoložive funkcije pločice ostvaruju se programiranjem mikrokontrolera čiji su priključci izvedeni na pločici i na koje se spajaju svi drugi elementi. Programiranje mikrokontrolera na Arduino pločici se vrši putem posebno razvijenog Arduino korisničkog programskog sučelja koje se koristi za programiranje svih Arduino razvojnih pločica. Za programiranje je potrebno spojiti pločicu na računalo putem USB priključka i na računalu pokrenuti Arduino korisničko sučelje za programiranje.

Slika 10. Osnovni dijelovi Arduino IDE sučelja za programiranje

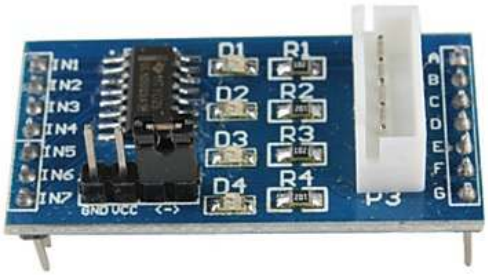
Slika 10 prikazuje izgled Arduino korisničkog sučelja za programiranje sa označenim osnovnim elementima sučelja. Alatna traka sadrži izbornike za postavljanje raznih opcija programiranja, odabir modela Arduino pločice i oznake USB priključka na koji je pločica spojena i dr. Na početku se pozivaju biblioteke za određene funkcije koje će se pozivati u programu. U dijelu postavki programa određuje se tip korištenih priključaka za program: digitalni ili analogni priključak, ulaz signala ili izlaz. Ovdje se još određuju i postavke serijske veze i brzine prijenosa podataka serijskom vezom te sve one funkcije koje se trebaju samo jednom izvršiti prilikom pokretanja programa.

U glavnoj petlji programa piše se kod koji će se nakon pokretanja programa izvršavati i obavljati neku željenu funkciju. Na serijskom monitoru se može, prilikom izvršavanja koda na mikrokontroleru, pratiti tijek odvijanja programa ili ispisivanje vrijednosti određenih varijabli čije promjene želimo pratiti. Nakon što je kod napisan potrebno je još provjeriti ispravnost sintakse i tek se onda kod može kompajlirati tj. prebaciti u oblik razumljiv mikrokontroleru te snimiti u memoriju mikrokontrolera. Pri dnu prozora sučelja nalazi se informacija o modelu priključene pločice i oznake korištenog USB priključka.

4.2. Motor driver shield

Zbog male izlazne snage koju na svom izlazu omogućuje Arduino pločica, ona će se koristiti samo kao logički dio upravljačkog sustava te će se njenim izlaznim signalima samo kontrolirati tok signala veće snage. Za osiguravanje primjene takvih signala potrebno je osigurati dodatni sklop koji će biti posrednik između Arduino pločice i motora.
Kao prvi izbor za upravljenje izabrana je raspoloživa pločica sa driverom za upravljanje pogonom jednog koračnog motora. Na ovoj pločici se nalazi IC ULN2003 driver za 4-fazni koračni motor, a koji je ustvari samo integrirana verzija osam tranzistora u jednom sklopu a koji se koristi kao digitalne sklopke za upravljanje fazama motora.

Na pločici se nalaze s lijeve strane ulazi sa mikrokontrolera koje se dalje spajaju na integrirani krug ULN2003 koji predstavlja polje tranzistora. Ovaj se sklop koristi uglavnom kao digitalno upravljane „sklopke“ za svaki kanal odnosno fazu koračnog motora. Upravljanjem „sklopki“ regulira se tok struje sa eksternog napajanja prema određenoj fazi motora. Određenim redoslijedom otvaranja i zatvaranja „sklopki“ povezano s time aktiviranja pojedinih namota, dobiva se efekt zakretanja rotora koračnog motora.



Slika 11. ULN2003 driver za upravljanje koračnim motorom

Iako je pločica prema propisanim karakteristikama trebala zadovoljavati ovu funkciju ipak se kroz testiranje pokazala kao neadekvatna i vrlo ograničavajuća u pogonu motora. Glavni uočeni nedostatak ove pločice je izlazna snaga motora a koja ovisi o maksimalnom napajanju pločice u iznosu 5 V. Svaki veći iznos napona napajanja mogao bi uništiti integrirani krug zbog velike disipacije snage na njemu te uzrokovanog prevelikog zagrijavanja elementa.

Ograničenje spajanja samo jednog motora na pločicu i potrebe za korištenjem minimalno 4 digitalna izlaza što u ovom zadatku doveli su u ovom slučaju do potrebe za 2 zasebne pločice i ukupno 8 digitalnih izlaza samo za pogon motora, ne uključujući ostale dijelove. Uz prethodno navedeno, za upravljanje radom ove pločice programski kod je dosta kompliciraniji nego što je to slučaj kod primjene posebnog shielda sa shift registrima za razne namijene od kojih je jedna i pogon koračnih motora.

Kao drugi [6], a kako se kasnije pokazalo i konačni izbor, izabrana je pločica 4x4 Driver Shield tvrtke Logos Electromechanical. Ova pločica na sebi ima dva 8-bitna shift registra 74AHCT595 i 4 IC MOSFET pojačala IPS6044. U ovoj kombinaciji tih elemenata se može pomoću shift registara kontrolirati 16 kanala i primjenom pojačala se mogu kontrolirati kanali sa iznosima struja do 5 A i naponima do 30 V. Pločica ima izvedene i priključke koji omogućavaju paralelno spajanje do 25 ovakvih pločica, čime je omogućena kontrola do 400 kanala.

Slika 12. 4x4 Motor driver shield [6]

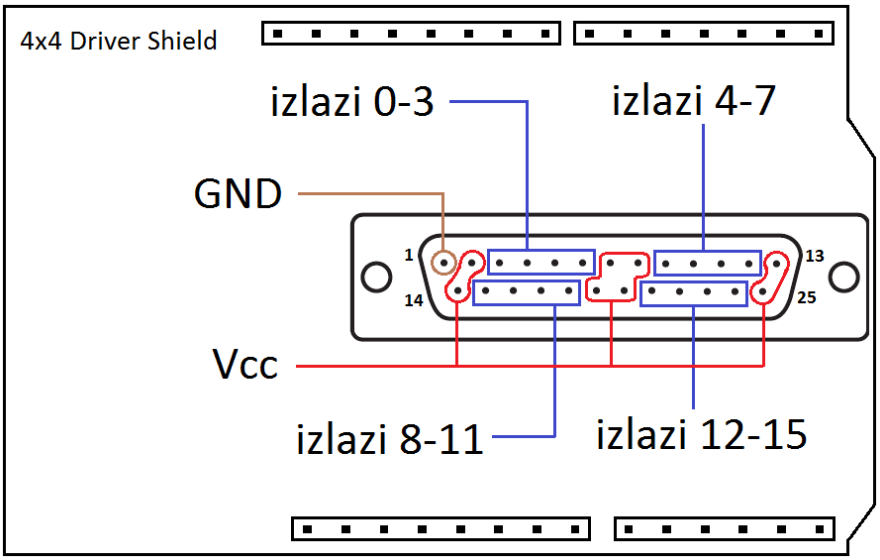
Nadalje, primjenom Arduino biblioteke za upravljanje shift registrima može se pomoću 4 izlaza sa Arduino pločice kontrolirati 16 i više izlaza na ovoj pločici. Mogućnost kontrole 16 izlaza može se iskoristiti i za upravljanje radom do četiri 4 - fazna koračna motora, što je ovdje u osnovi i glavna ideja primjene ovog shield-a.

Tablica 3. Korišteni priključci Arduino pločice za upravljanje motor shieldom

Redni broj digitalnog izlaza	Funkcija
7	Latch pin
8	Master reset pin
11	Serial input (data)
13	Serial clock

Pločica se spaja izravno na Arduino pločicu poklapanjem priključaka jedne i druge pločice te umetanjem priključaka sa shield-a u utore na Arduino pločici. Dio sa shift registrima i arduinom predstavlja logički dio upravljanja sustava i on radi sa standardnom digitalnom TTL razinom napona 5V. Drugi dio upravljačkog sustava čine MOSFET pojačala

koja upravljačke TTL signale pomoću zasebnog napajanja driver shielda pretvaraju u upravljačke signale sa dovoljnom razinom energije za pogon motora. Izlazi sa driver shielda su izvedeni preko paralelnog DB25 muškog priključka koji sadrži navedenih 16 kontrolabilnih izlaza (4 skupine po 4 izlaza), 8 priključaka za napajanje Vcc (6 – 30V) i 1 priključka zajedničkog uzemljenja (GND) za sve izlaze. Shematski tlocrt shielda i raspored svih priključaka na njemu je prikazan u nastavku (Slika 13).



Slika 13. Raspored priključaka na motor driver shield-u

4.3. Upravljanje rada motora [7]

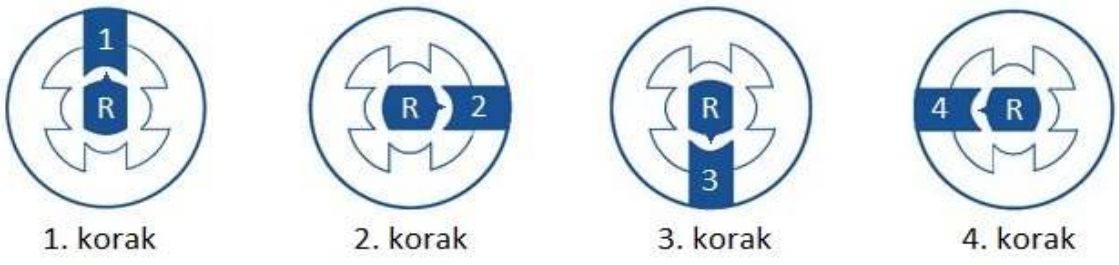
Koračni motori korišteni za pogon u ovom mehanizmu robotizirane glave mogu biti pokretani u 4 moguća režima rada i to:

- tzv. „valni“ režim ili aktiviranjem jedne faze
- aktivacijom dvije faze
- kombiniranom aktivacijom jedne ili dvije faze
- mikrokoračni režim

Svaki režim će biti opisan principno, a u praksi se za upravljanje najviše koristi neki od prva tri navedena režima rada.

4.3.1. Režim aktiviranjem jedne faze

Valni režim rada, kako se još naziva, opisuje metodu pokretanja koračnog motora u kojoj se za svaki korak okretanja rotora aktivira samo jedna faza motora. Sa ovom metodom se rotor motora prilikom svakog koraka zakrene za 1,8°. Svaki sljedeći korak odvija se aktivacijom sljedeće faze te se tako faze aktiviraju slijedno u kružnom ciklusu i od tu dolazi naziv „valni“ režim. Prilikom uključivanja jedne faze ona se elektrizira kao južni pol te privlačenjem sjevernog pola magneta rotora zakreće rotor. Svaka aktivacija iduće faze čini istu stvar te tako zakreću rotor koračnog motora. Ilustracija ove metode prikazana je na slici ispod (**Slika 14**). (Simbolično je ovdje zakret od 90° koji predstavlja jedan korak – 1,8°)



Slika 14. Režim aktiviranje jedne faze motora

4.3.2. Režim aktiviranjem dviju faza

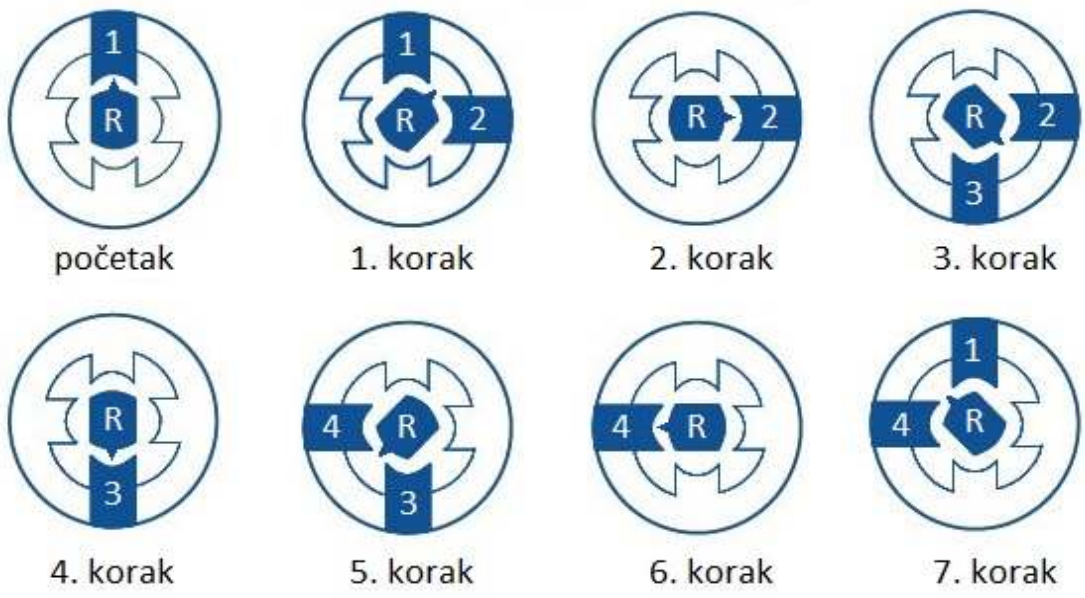
Kod metode aktivacije dvije faze se u isto vrijeme aktiviraju dvije faze i za svaki sljedeći korak aktiviraju se slijedeće dvije faze redoslijedom. Kod ove metode se rotor, isto kao kod prve metode, u jednom koraku zakrene za 1,8° a razlika je jedino u većem momentu koji je kod ove metode jednak rezultanti od dva momenta nastalih djelovanjem dviju faza.

Primjerice u slučaju faza zakrenutih za 90° rezultantni moment ima iznos 1,4 puta veći od iznosa momenta kod jedne faze. Zbog navedenih prednosti ove metode a ujedno i jednostavnosti izvedbe iste, ona je izabrana za korištenje prilikom pogona motora na postolju glave robota. Ilustracija ove metode prikazana je na slici ispod (**Slika 15**).

Slika 15. Režim rada motora aktivacijom dviju faza

4.3.3. Kombinirani režim

Ova metoda je kombinacija dviju prethodno opisanih metoda. U prvom koraku kod ove metode se aktivira samo jedna faza te se rotor poravna sa tom fazom. U sljedećem koraku se uz aktiviranu prvu fazu aktivira i druga faza u nizu te se rotor motora poravna između tih dviju faza. U sljedećem koraku je uključena samo druga faza koja svojim djelovanjem zakreće i poravnava rotor motora sa drugom fazom. Sljedeći korak je da se uz uključenu drugu fazu uključi i treća faza i tako dalje u krug. Ovom se metodom u svakom koraku postiže zakret u iznosu pola vrijednosti punog koraka tj. zakret rotora u svakom koraku iznosi 0,9°.

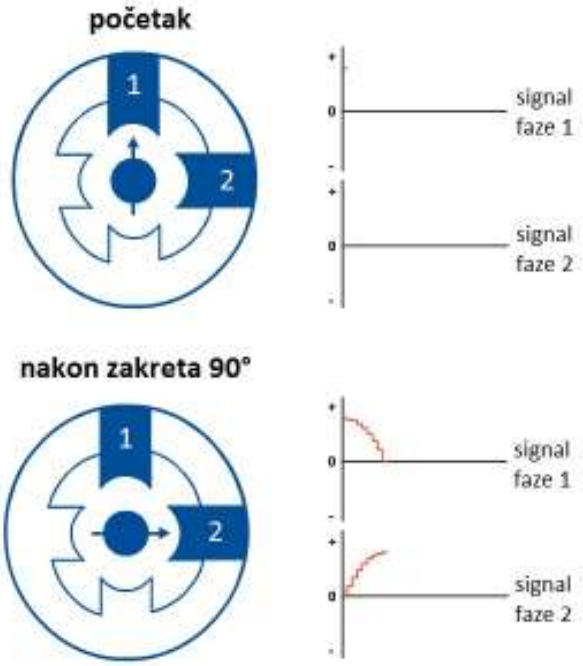


Slika 16. Kombinirani režim rada motora

4.3.4. Mikrokoračni režim

U mikrokoračnom režimu rada, za razliku od svih dosad navedenih metoda, faze se ne nalaze samo u krajnjim stanjima uključenosti ili isključenosti, već se one djelomično uključuju odnosno isključuju. To se postiže tako što se vrijednost upravljačkog signala za aktivaciju faze stepenasto mijenja u vremenu. Primjerice, na početku gibanja rotor je poravnat sa prvom fazom koja je uključena sa maksimalnom vrijednosti upravljačkog signala dok se u tom trenutku druga faza nalazi potpuno isključena.

Stepenastim smanjenjem vrijednosti signala na prvoj fazi i istovremenim povećavanjem vrijednosti signala za uključivanje druge faze se odvija zakretanje rotora uslijed promjene iznosa momenta na fazama i kuta rezultantnog momenta između tih dviju faza.



Slika 17. Mikrokoračni režim rada motora

4.3.5. Programsko upravljanje radom motora

Svi prethodno opisani režimi predstavljeni su fizičkim opisom promjena stanja prilikom zakreta rotora motora. Da bi se to tako odvijalo brine se upravljačka jedinica čije se funkcije ostvaruju prema određenom programu, koji indirektno predstavlja određeni režim rada.

Rad motora u svakom režimu prema tome upravlja se programski, kontrolirajući stanje izlaznih vrijednosti na pločici mikrokontrolera. Kontrola se vrši na logičkoj TTL razini koristeći digitalne izlaze na pločici mikrokontrolera tako da izlazni signali imaju digitalne vrijednosti: logička „0“ za neaktivno stanje tj. na izlazu je 0 V i logički „1“ za aktivno stanje tj. na izlazu je vrijednost 5 V.

Kao što je ranije navedeno u jednom od prethodnih poglavlja, prva opcija za upravljanje je bila primjena pločice sa integriranim driver-om ULN2003 za pogon koračnog motora, ulazima sa mikrokontrolera te izlazima za spoj na motor. Korištena pločica nepoznatog proizvođača, prikazana je u poglavlju o motor driver shieldu(**Slika 12**).

Kao drugi izbor navedena je primjena 4x4 driver shielda za spajanje na Arduino pločicu i koji je omogućavao pogon koračnih motora sa većim iznosom upravljačkih signala.

Programiranje rada motora u slučaju primjene prve pločice se može izvesti na dva načina. Prvi način je pisanje programa u Arduino okruženju tako da se upravlja stanjima svih upravljačkih izlaza za kontrolu faza motora, sekvencom njihovih izmjena – indirektno smjerom okretanja rotora motora, vremenom između pojedinih koraka – indirektno brzina okretanja rotora motora te brojem promjena stanja izlaza – indirektno brojem koraka motora.

Drugi, dosta jednostavniji način je primjenom postojeće Arduino Stepper biblioteke u kojoj su unaprijed definirane funkcije koje reguliraju stanja izlaza ovisno o izabranim parametrima brzine, smjera i broja koraka. Paralelna usporedba kompleksnosti programskog koda za primjenu prve pločice sa ULN2003 driverom, za identični zadatak okretanja rotora u jednu stranu za puni krug, za oba načina dana je u tablici.

Tablica 4. Usporedba programskog koda primjenom dvije različite metode

Kontrolom stanja izlaza	Primjenom Stepper biblioteke

int faza1 = 8;
int faza2 = 9;
int faza3 = 10;
int faza4 = 11;
int pauza = 500;
int i;
...
 	
	#include <Stepper.h>
	#define STEPS 200
	Stepper stepper(STEPS, 5, 7, 6, 4);
	
	...

```
void setup() {
  pinMode(faza1, OUTPUT);
  pinMode(faza2, OUTPUT);
  pinMode(faza3, OUTPUT);
  pinMode(faza4, OUTPUT);

for ( i = 0 ; i < 50 ; i++) {
  digitalWrite(faza1, HIGH);
  digitalWrite(faza2, HIGH);
  digitalWrite(faza3, LOW);
  digitalWrite(faza4, LOW);
  delay(pauza);

  digitalWrite(faza1, LOW);
  digitalWrite(faza2, HIGH);
  digitalWrite(faza3, HIGH);
  digitalWrite(faza4, LOW);
  delay(pauza);

  digitalWrite(faza1, LOW);
  digitalWrite(faza2, LOW);
  digitalWrite(faza3, HIGH);
  digitalWrite(faza4, HIGH);
  delay(pauza);

  digitalWrite(faza1, HIGH);
  digitalWrite(faza2, LOW);
  digitalWrite(faza3, LOW);
  digitalWrite(faza4, HIGH);
  delay(pauza);
}
}
void loop()
{
}
```

```
void setup()
{
  stepper.setSpeed(30);
  delay(1000);
  stepper.step(200);
  delay(1000);

  digitalWrite(4,LOW);
  digitalWrite(5,LOW);
  digitalWrite(6,LOW);
  digitalWrite(7,LOW);
}
void loop()
{
}
```

Primjenom druge pločice, 4x4 driver shielda, zbog karakteristika i shift registara u sklopu pločice upravljanje je izvedeno višestruko jednostavnije nego u prvotnom slučaju. Kod ove pločice je omogućeno da se korištenjem samo 4 izlaza sa upravljačke jedinice upravlja radom driver shielda te kontrolira do 4 koračna motora istovremeno. Ovo je ostvareno uz primjenu Arduino biblioteke za rad sa shift registrima koja uvelike olakšava pisanje programa za upravljanje radom motora. Korištenjem funkcija iz ove Arduino biblioteke naredbom shiftOut kontroliraju se stanja 8 izlaza tj. povezano s time, 8 faza motora odnosno 2 koračna motora. Naredba se poziva na slijedeći način:

```
shiftOut(11, 13, MSBFIRST, lowByte)
```
(3)

gdje prvi broj u zagradi, 11, označava priključak na shield „Serial input“ tj. kanal za podatke, drugi broj, 13, označava priključak na shield „Serial clock“ tj. kanal takta za sinkronizaciju prijenosa podataka, oznaka MSBFIRST (od eng. Most Significant Bit - FIRST) označava slanje prvog elementa sa najznačajnijim utjecajem tj. kod binarnih brojeva je to prvi element s lijeve strane (element sa najvećom potencijom) te na kraju lowByte označava kontrolu izlaza označenih brojevima 0-7.

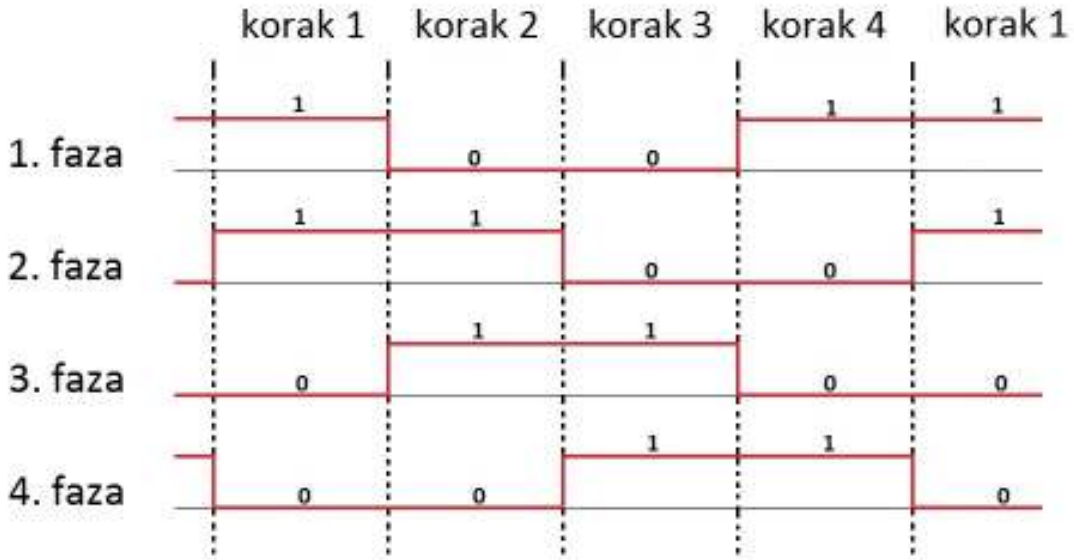
Ukoliko se želi slati prvi element sa najmanjim utjecajem onda se u naredbi označava LSBFIRST (od eng. Least Significant Bit - FIRST) i za kontrolu izlaza označenih brojevima 8-15 se u naredbi označava na kraju highByte umjesto lowByte.
Primjer programskog koda za upravljenje driver shieldom primjenom Arduino biblioteke za shift registre prikazan je u nastavku.

Tablica 5. Primjer programskog koda korištenjem biblioteke za shift registre

```
int latchPin = 7;
int clockPin = 13;
int dataPin = 11;

void setup() {
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}
void loop()  {
  // 1. korak
  digitalWrite(latchPin, LOW);
  digitalWrite(8, HIGH);
  shiftOut(dataPin, clockPin, MSBFIRST, 3);
  digitalWrite(latchPin, HIGH);
  delay(dt); i--;
  // 2. korak
  digitalWrite(latchPin, LOW);
  digitalWrite(8, HIGH);
  shiftOut(dataPin, clockPin, MSBFIRST, 6);
  digitalWrite(latchPin, HIGH);
  delay(dt); i--;
 // 3. korak
  digitalWrite(latchPin, LOW);
  digitalWrite(8, HIGH);
  shiftOut(dataPin, clockPin, MSBFIRST, 12);
  digitalWrite(latchPin, HIGH);
  delay(dt); i--;
 // 4. korak
  digitalWrite(latchPin, LOW);
  digitalWrite(8, HIGH);
  shiftOut(dataPin, clockPin, MSBFIRST, 3);
  digitalWrite(latchPin, HIGH);
  delay(dt); i--;
}
```

Za upravljanje je odabran režim aktivacije dviju faza istovremeno kako bi se dobio što veći iznos momenta motora za pogon mehanizma. Vidljivo u prikazanom programskom kodu, u naredbi shiftOut se na kraju zapisuju 4 dekadska broja: 3, 6, 12 i 9, a ovaj broj vrijednosti povezan je s brojem faza motora. Svaki od ovih brojeva predstavlja jedan korak zakreta motora prema izabranom režimu.



Slika 18. Dijagrami promjene signala po fazama motora

Usporedno gledajući programski kod i promjene signala (**Slika 18**) može se vidjeti poveznica između sekvence aktivacije pojedine faze i naredbe za izvršavanje određenog koraka zakreta. U programskom kodu za prvi korak zakreta motora u naredbi je zapisan dekadski broj 3 koji u binarnom zapisu ima vrijednost 0011 a koji se u isto vrijeme može uočiti gledajući vrijednosti signala svih faza u prvom koraku, odozdo prema gore. Isto vrijedi i za ostale, u drugom koraku u naredbi je broj 6 – binarni zapis 0110 i tako slijedno za ostale korake. Nakon četvrtog koraka sekvenca se opet ponavlja istim redoslijedom.

Ovisno o željenom smjeru zakretanja motora će ovisit sekvenca odvijanja koraka, za ovdje prikazan smjer je sekvenca 1-2-3-4 dok je za obrnuti smjer sekvenca 1-4-3-2, odnosno kao da se dijagram aktivacije faza po koracima odvija gledajući u lijevu stranu.

5. MICROSOFT KINECT

Jedan od bitnih dijelova glave zamišljenog humanoidnog robota su njegove oči putem kojih bi robot primao informacije o stanju okoline. U ovom slučaju za oči robota predviđeno je korištenje Microsoftovog Kinect uređaja čijim bi se korištenjem, zbog njegovih raznih pogodnosti, znatno proširila početno predviđena funkcija primanja osnovnih informacija iz okoline robota.

5.1. Općenito o Kinect senzoru [8]

Uređaj je u početku zamišljen kao dodatni periferni uređaj namijenjen korisnicima Xbox 360 igraće konzole, kako bi dodatno proširili mogućnosti igrivosti te tako postigli neku novu razinu među igraćim konzolama. Ovaj model pušten je u prodaju u studenom 2010. Nedugo nakon početnog „igračkog“ uvoda, objavljen je dolazak novog modela nazvanog „Kinect for Windows“, što je i učinjeno u veljači 2012. puštanjem u prodaju najavljenog modela namijenjenog posebno za razvoj aplikacija na Microsoft Windows platformi.

Iako su ova dva modela bila vizualno gotovo identična a po tehničkim karakteristikama prilično slični ipak su postojale sotverske razlike koje su dosta izdizale model za Windows platformu od onoga za Xbox konzolu. Naime model za Windows platformu je dolazio sa 6 softverskih značajki koje su uvelike bile orijentirane za razvoj aplikacija sa Kinect-om.

Te značajke su slijedeće:
1. tzv. „Near mode“ – omogućava rad senzora na udaljenostima već od 40 cm pa sve do 3 m udaljenosti, bez smanjenja preciznosti prilikom rada
2. U aplikaciji praćenja skeleta (Skeletal tracking) omogućava praćenje skeleta korisnika u sjedećem položaju odnosno aplikacija prati samo 10 gornjih točaka skeleta
3. Proširene mogućnosti postavki video kamere poput svijetline, ekspozicije i sl.
4. tzv. „Handgrip“ – funkcija koja prepoznaje gibanje prstiju te omogućava nove funkcije poput povećanja prikaza slike pri pomaku prstiju ili hvatanje elemenata također prilikom pomicanja prstiju ruke

5. Mogućnost korištenja funkcije „Kinect Fusion“ za mapiranje okoline u 3D tehnici ili uređivanje mapiranih elemenata
6. USB priključak - za olakšano korištenje na računalu i drugim USB podržanim uređajima

Slika 19. Kinect za Xbox – model 1.0

Uz sami uređaj, korisnicima je na raspolaganje dano i razvojno okruženje „Kinect for Windows SDK“ za lakše upoznavanje sa radom uređaja i za lakši razvoj aplikacija korištenjem Kinect-a. Aplikacije se mogu napraviti pisanjem programskog koda u kojem se pozivaju i koriste sve navedene funkcije Kinect-a. Programski jezici za pisanje koda i korištenje funkcija Kinect-a za izradu aplikacija su C++, C# (C sharp) i Visual Basic i svi su dostupni za korištenje u Microsoft Visual Studio okruženju.

U razvojnom okruženju „Kinect for Windows SDK“ se nalaze dvije programske aplikacije: „Kinect Studio“ i „Developer Toolkit“. Aplikacija „Kinect Studio“ služi za jednostavno povezivanje Kinect-a sa računalom i pristupa njegovim funkcijama poput pregleda i snimanja zapisa sa RGB i dubinske kamere kao i korištenje ugrađenih mikrofona.

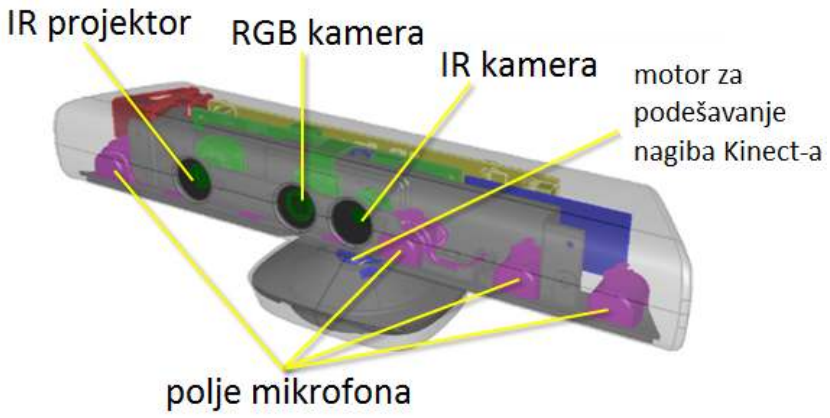
Druga aplikacija „Developer Toolkit“ objedinjuje razne primjere programskih kodova napisanih u C++, C# ili Visual Basic-u, a koji u kombinaciji sa Kinect-om obavljaju neku funkciju (primjerice prepoznavanje boja ili oblika sa slike, prepoznavanje skeleta ili lica korisnika i sl.). Daljnjim razvojem u slijedećim godinama uređaj je izmijenjen tehnički u vidu poboljšanja karakteristika i performansi kao i vizualno, te je u 2014. pušten u prodaju model „Kinect for Windows v2“.

Slika 20. Kinect za Windows platformu - model 2.0

Za korištenje u ovoj primjeni, za glavu humanoidnog robota, odabran je raspoloživi i do sada dobro upoznati prvi model Kinect-a za Windows platformu (**Slika 19**) o kojem će se detaljnije karakteristike dati u slijedećem poglavlju.

5.2. Tehničke karakteristike Kinect senzora

Uređaj ima ugrađene video kameru u boji, tzv. dubinsku kameru koja se sastoji od infracrvenog (IR) projektora i infracrvene (IR) kamere te polja od 4 mikrofona postavljenih uzduž cijelog kućišta Kinect senzora. Interakcija korisnika sa računalom se postiže koristeći ove elemente i njihove programabilne funkcije kao što su detekcija korisnika, praćenje pokreta, prepoznavanje glasa i sl. Time se omogućava da korisnik upravlja radom uređaja i raznih aplikacija kroz prirodno inteligentno sučelje što do sada nije bio slučaj.



Slika 21. Osnovni elementi Kinect senzora

5.2.1. Video kamera

Kinect ima ugrađenu video kameru, koja omogućava snimanje u boji u rezoluciji 640 x 480 piksela. Maksimalna kvaliteta slike sa kamere se može dobiti u 32-bitnom prikazu boja što je oko 4,3 mld. boja (4 294 967 296 točan broj boja u 32-bitnom prikazu boja) pri frekvenciji prikaza od 30 fps (od eng. frame per second što označava broj slika prikazan u jednoj sekundi). Nadalje, vidno polje kamere iznosi otprilike 62° horizontalno i 48° vertikalno što daje razlučivost prikaza od otprilike 10 x 10 piksela po stupnju. Ova kamera se može koristiti za spremanje slika i video zapisa prilikom rada sa uređajem a uz implementirane algoritme za obradu slike i u kombinaciji sa dubinskom kamerom koristi se za razne primjene prepoznavanja skeleta ili kontura lica korisnika, praćenje pokreta i sl.

Slika 22. Prikaz slike sa video kamere Kinect-a

5.2.2. Dubinska kamera

Na Kinect-u je dubinska kamera izvedena kao kombinacija dvaju elemenata: infracrvenoga projektora i infracrvene kamere. Pomoću projektora se sa Kinect-a u njegovo vidno polje projicira točkasti uzorak laserskih zraka. Tehnika određivanja dubine zasniva se na principu tzv. „strukturnog svjetla“ kod kojega se projicira poznati uzorak svjetlosti na neku

površinu te se potom taj uzorak promatra kamerom. Iz odstupanja promatranog uzorka od poznatog projiciranog uzorka, putem raznih matematičkih algoritama i izračuna, dobije se kontura elementa iz radnog prostora te njegova prostorna udaljenost od samog Kinect-a.

Slika 23. Projekcija infracrvenog uzorka sa Kinect-ove dubinske kamere

U slučaju Kinect-a korištenje infracrvenog laserskog svjetla omogućava prilično precizno određivanje dubine elemenata na slici i u prilikama raznih načina i intenziteta osvjetljenja promatranog prostora, iako u malo tamnijem prostoru Kinect daje ponešto bolje rezultate. Nadalje, razlučivost infracrvene kamere iznosi maksimalno 320 x 240 piksela dok joj vidno polje iznosi približno 58° u horizontalnom smjeru i približno 46° u vertikalnom smjeru što nam daje približan omjer od 5 x 5 piksela po stupnju vidnog polja kamere, što je zadovoljavajuće za ovu primjenu.

Svaki piksel iz snimljene mape udaljenosti se zamjenjuje sa pripadajućim pikselom koji sadrži informaciju o udaljenosti u obliku nijanse sive boje (eng. grayscale). Tako se pikseli elemenata slike koji se nalaze bliže objektivu senzora zamjenjuju pikselima svjetlije nijanse (nijansa najbližih elemenata se približava bijeloj boji) dok su udaljeniji elementi tamnije nijanse (najudaljeniji elementi poprimaju piksele sa nijansama crne boje).

Slika 24. Prikaz mape dubine sa Kinect-a

Rad Kinect senzora je ograničen, kako fizičkom izvedbom tako i softverskim dijelom. Ranije je u tekstu spomenuto postojanje radnog prostora Kinect senzora tj. prostora u kojem Kinect senzor može obavljati svoje funkcije praćenja, prepoznavanja i sl., a sve izvan tog prostora je za Kinect „nevidljivo“ odnosno senzor u tom prostoru neće pokazivati ispravne rezultate. Model Kinect-a za Xbox ima normalno radno područje od 0,8 m do 4 m a model za Windows platformu, primjenom značajke „Near mode“ za rad u blizini te time ima radno područje od 0,4 m do 3 m. **Slika 25** ilustrira radno područje za modele Kinect-a za Xbox360 i za Windows platformu.

Slika 25. Usporedba radnog područja Kinect senzora za Xbox i Windows

5.2.3. Mikrofoni

U Kinect je ugrađeno i tzv. „polje“ mikrofona, točnije njih četiri a raspoređeni su uzduž kućišta senzora. Primjena ovakvog sustava sa više mikrofona omogućuje veću kvalitetu zvučnog zapisa a programskom obradom signala ostvaruje se sposobnost prepoznavanja glasa i upravljanje pomoću glasovnih naredbi. Uz četiri mikrofona u sustavu za obradu zvuka se nalazi i posebni elektronički sklop za obradu signala sa mikrofona koji razdvaja svaki signal posebno te tako omogućava određivanje smjera iz kojeg dolazi stvarni zvuk korisnika dok se pritome vrlo elegantno može ukloniti buka ili jeka koji dolaze iz korisnikove okoline a nisu poželjni za daljnju obradu.

5.3. Primjena

Početna primjena modela za Xbox bila je isključivo u igračke svrhe jer je ovaj model Kinect-a prvenstveno i bio namjenjen korisnicima igraće konzole. Iako je Kinect imao neka ograničenja u radu, zbog svojih je karakteristika i skupa senzora koje je sadržavao odmah zamijećen i u krugovima programera i raznih hobista koji su se bavili razvojem raznih aplikacija koje su uključivale slične senzore i funkcije kakve Kinect posjeduje.

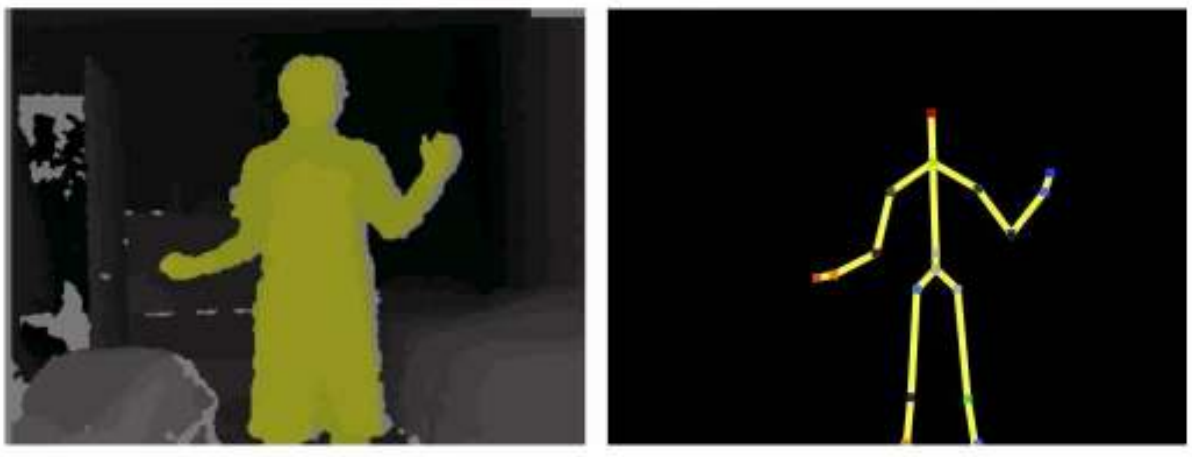
Nakon izlaska modela za Windows platformu znatno su se povećale mogućnosti primjene za svrhe razvoja aplikacija te se Kinect počeo brzo širiti i primjenjivati u računalstvu i robotici. Kinect je zbog tehničkih karakteristika poput kamera za snimanje slike i obradu te za određivanje udaljenosti te polja mikrofona i softverske platforme koja omogućava raznolike funkcije prepoznavanja korisnika, praćenje kretnji i gesta, upravljanje glasom i dr., postao često primjenjivani element u mobilnoj robotici i u zadacima navođenja industrijskih robota.

Slika 26. Primjer primjene Kinect senzora u mobilnoj robotici

5.4. Aplikacija praćenja skeleta [14]

Jedan od zadataka u radu je napraviti gibanje glave humanoidnog robota u ovisnosti o položaju korisnika u prostoru tako da je tablet tj. „lice“ glave robota usmjereno prema glavi korisnika koji se nalazi u promatranom radnom području kako bi se ostvarila funkcija praćenja korisnika pogledom. Za ostvarivanje ovog zadatka u početku je korištena fukcija prepoznavanja kontura lica (eng. Face tracking) no kasnije se pokazalo kako ova funkcija radi stabilno uglavnom pri malim udaljenostima što se za ovu primjenu nije baš pokazalo kao najboljom opcijom. Druga okolnost ove funkcije koja nije išla u korist ove primjene je koordinatni sustav lica koje se gotovo nikako ne može međusobno povezati sa koordinatnim sustavom glave, pogotovo zbog ograničenog opsega gibanja glave dok korisnik s druge strane može nesmetano pomicati lice u prostoru. Iz ovog je razloga ovdje odlučeno koristiti funkciju praćenja skeleta (eng. Skeletal tracking) u kojoj se prati skelet korisnika u promatranom prostoru te se glava korisnika također prati kao jedna točka u prostoru što je ovdje dobra pogodnost.

Funkcija nam daje koordinate svih 20 točaka skeleta te je jednostavno pristupiti vrijednostima svake pojedine točke kako bi dobili željene koordinate u X, Y i Z smjeru gdje se vrijednost koordinate u smjeru Z osi ustvari dobije iz mape dubine opisane u jednom od prethodnih poglavlja.



Slika 27. Funkcija praćenja cijelog skeleta(desno) uz prikaz dubinske slike(lijevo)

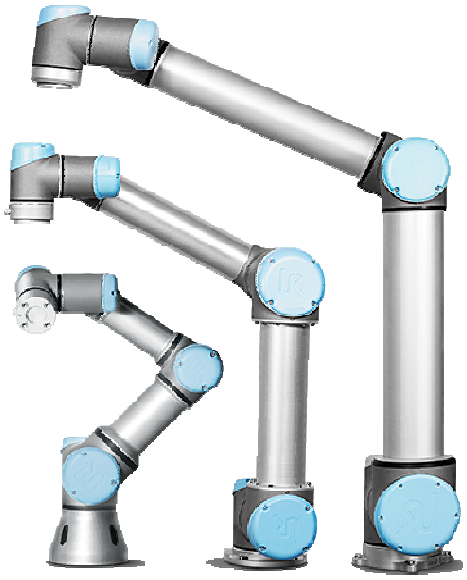
6. ROBOTSKA RUKA [10]

Drugi dio rada obuhvaća povezivanje komercijalne industrijske robotske ruke sa 6 stupnjeva slobode gibanja i postolja glave humanoidnog robota kako bi se ostvarilo kooperativno djelovanje ta dva elementa. Ideja za ostvarivanje ove kombinacije je došla iz čovjekovog prirodnog djelovanja tj. njegovog instinktivnog djelovanja prilikom obavljanja neke zadaće ili jednostavno u skakodnevnim radnjama koje obavlja. Temelj ideje je u vizualnom praćenju obavljanja neke zadaće odnosno rukovanja određenim elementima rada.

Neuobičajeni pokreti ljudske ruke tj. pokreti koji se ne vrše podsvjesno kao i pokreti koji uključuju kretnje u nepoznatoj okolini, sa nepoznatim elementima u njoj, zahtijevaju vizualno ispitivanje tj. svaka se takva kretnja kod ljudi instinktivno prati pogledom. Nakon prvotnog ispitivanja sadržaja okoline i ukoliko se okolina ne mijenja, moguće je obavljati neke radnje i funkcije bez daljnjeg vizualnog kontakta sa tim radnim područjem. Upravo se na ovom principu temelji gibanje postolja glave humanoidnog robota u kooperaciji sa radom robotske ruke. Sustav dobiva od robotske ruke informaciju o točnom položaju vrha alata robotske ruke te potom ovisno o njoj šalje upravljačke signale motorima te vrši orijentiranje postolja glave humanoidnog robota tj. usmjerava „pogled“ glave u smjeru položaja vrha alata robotske ruke.

6.1. Općenito o robotskoj ruci

Robotska ruka je izabrana prema raspoloživosti i to od proizvođača Universal Robots, model robotske ruke je UR5. Tvrtka Universal Robots je, sa približno 10 godina postojanja, relativno novo ime ne tržištu industrijskih robota. U proteklih 6 godina rada razvili su 3 modela robotske ruke: 2009.g. UR5 – nosivosti do 5 kg, 2012.g. UR10 – nosivosti do 10 kg te 2015.g. UR3 – nosivosti do 3 kg. Svi modeli razvijeni su s idejom da budu što je moguće manje mase, da ih je lako postaviti i programirati te su u početku bili prvenstveno zamišljeni za korištenje u prehrambenoj industriji.



Slika 28. Tri modela iz ponude Universal Robot-a

6.2. Tehničke karakteristike - UR5

Model UR5, kao i ostala dva modela, je industrijski robot sa 6 rotacijskih stupnjeva slobode gibanja, omogućava jednostavno i brzo postavljanje. Sastoji se od kombinacije dijelova izrađenih od aluminija i od polimera što ga čini laganim osiguravajući mu istovremeno dosta visoku nosivost. Nakon puštanja u rad potrebno je malo vremena dok se korisnik privikne na programsko sučelje te vrlo brzo postane vješt u programiranju gibanja robota čak i ako do tada nije imao puno kontakta sa programiranjem. Ostali tehnički podaci mogu se naći u tablici na slijedećoj strani (**Tablica 6**).

Tablica 6. Tehničke karakteristike robotske ruke

Broj stupnjeva slobode gibanja	6
Masa	18,4 kg
Nosivost	5 kg
Radni doseg	850 mm
Radni opseg zglobova	± 360°
Brzina zglobova (svi zglobovi)	180°/s
Brzina alata	1 m/s
Ponovljivost	± 0,1 mm
Snaga	≈ 200 W
Napajanje	100 – 240 VAC, 50-60 Hz
Temperaturno radno područje	0 – 50°C

6.2.1. Upravljačka jedinica

Sustav robotske ruke je jednostavan, sastoji se od robotske ruke, pripadajuće upravljačke jedinice te privjeska za učenje i programiranje. Na upravljačku jedinicu se spajaju kabel sa robotske ruke, kabel za napajanje cijelog sustava te privjesak za učenje i programiranje.

Slika 29. Upravljačka jedinica robotske ruke sa privjeskom za učenje

U upravljačkoj jedinici se uz sklop za napajanje nalaze i razni moduli za spajanje drugih elemenata na robota (razni senzori, aktuatori i sl.) te moduli za komunikaciju također sa ostalim elementima u industriji koristeći razne protokole poput MODBUS-a i sl. Osim ulazno-izlaznih modula, smještenih u upravljačkoj jedinici, na robotskoj se ruci još nalazi na zadnjem članku konektor za alat, a koji se sastoji od 8 priključaka i to: 2 digitalna ulaza, 2 digitalna izlaza, 2 analogna ulaza, priključka za napajanje +12 V/+24 V i uzemljenja. Popis priključaka koji se nalaze u upravljačkoj jedinici i ostalih specifikacija, prikazan je u nastavku. (**Error! Reference source not found.**)

Tablica 7. Karakteristike upravljačke jedinice robotske ruke

Dimenzije upravljačke jedinice (DxVxŠ)		475 mm x 423 mm x 268 mm
Napajanje	Upravljačke jedinice	24 V, 2 A
	Alata	12/24 V, 600 mA
Broj ulaza/izlaza	Digitalnih ulaza	16
	Digitalnih izlaza	16
	Analognih ulaza	2
	Analognih izlaza	2
Komunikacijski priključak		TCP/IP 100 Mb/s

6.3. Primjena

Početna ideja namjene robota prilikom razvoja ove linije industrijskih robota bila je za prehrambenu industriju ali se nakon razvoja uglavnom primjenjuje u raznim drugim industrijskim granama i primjenama.

Pa se tako danas ovi roboti mogu pronaći u slijedećim industrijskim aplikacijama:

- klasični „pick and place“ zadatak
- pakiranje proizvoda i paletizacija
- ljepljenje i zavarivanje elemenata
- montaža elemenata
- kontrola kvalitete u proizvodnji
- kooperacija u procesu injekcijskog prešanja
- posluživanje CNC strojeva i dr.

Zbog jednostavnosti, pristupačnosti za programiranje te prihvatljive cijene dosta su primjenjivani i u edukacijskim krugovima. Primjena ove robotske ruke u kooperaciji sa raznim drugim senzorima i aktuatorima u svrhu dobivanja novog inteligentnijeg sustava biti će prikazana kroz ovaj rad.

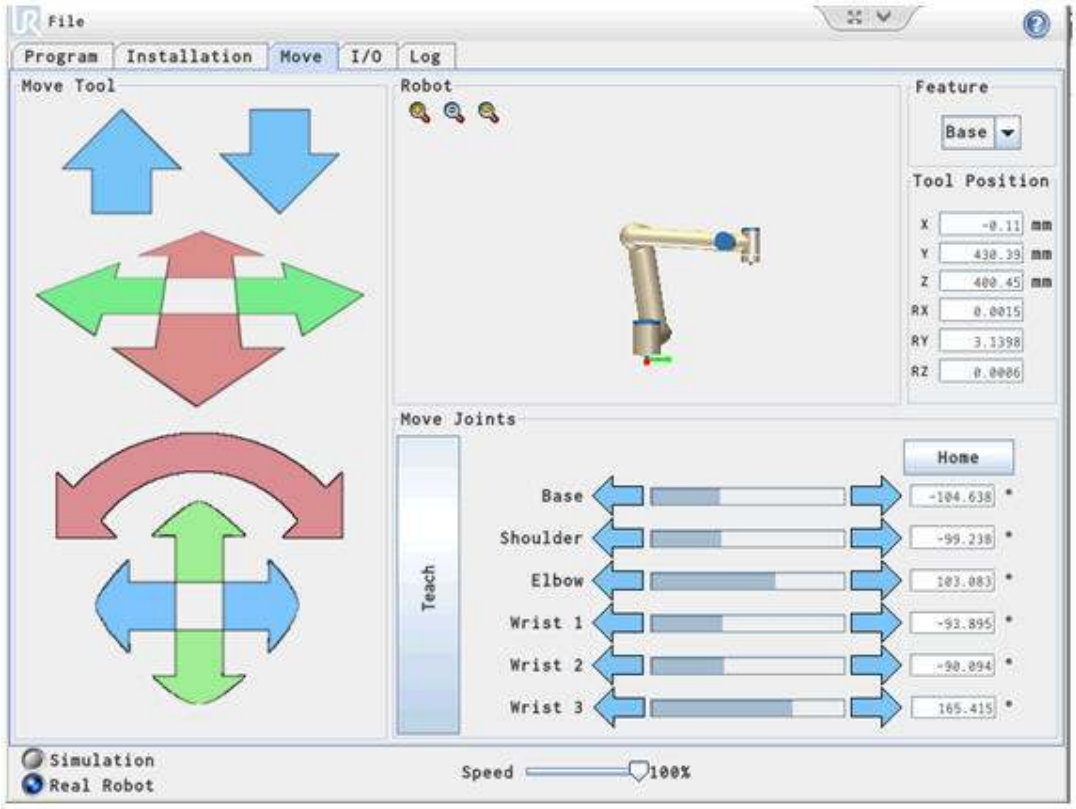
6.4. Programiranje

Kako je spomenuto na početku ovog poglavlja, robotska ruka UR5 se jednostavno i brzo može programirati i to na dva načina: pomoću privjeska za učenje i programiranje te putem API sučelja (od eng. Application Programming Interface – aplikacijsko programsko sučelje). Razlika između njih je u načinu programiranja gibanja robota i pristupa robotu tj. ova podjela se temelji na „online“ i „offline“ principu programiranja.

6.4.1. Privjesak za učenje

Korištenje privjeska za učenje i programiranje predstavlja „online“ način programiranja robota i ne zahtjeva nikakve druge elemente osim osnovnih elemenata robotskog sustava nevedenih u jednom od prethodnih poglavlja. Ova metoda je najjednostavnija i najbrža metoda a ujedno i najpristupačnija za korisnika, pogotovo ukoliko korisnik nije vješt u programiranju.

Privjesak je izveden kao 12 inčni ekran osjetljiv na dodir, pokraj ekrana se nalazi sigurnosno tipkalo za zaustavljanje rada robota, a s donje strane se nalazi tipkalo za sigurno pokretanje robota i USB utor. Izgled grafičkog sučelje sa privjeska, PolyScope-a, vidljivo je na slici ispod.



Slika 30. Izgled grafičkog sučelja na privjesku za učenje

U ovom načinu programiranja korisnik jednostavno „slaže“ program koristeći definirane naredbe za linearno ili kružno gibanje vrha alata, zakretanje pojedinih zglobova, preddefinirane funkcije i petlje, korištenje ulaza i izlaza iz upravljačke jedinice ili alata, zapisivanje i čitanje podataka iz registara i sl. Velika pogodnost je u ovom načinu i mogućnost korištenja tzv. „teach“ principa učenja pozicije, pri kojemu korisnik može brzo i jednostavno vlastitom rukom zakretati sve zglobove robota i tako ga dovesti u željenu orijentaciju.

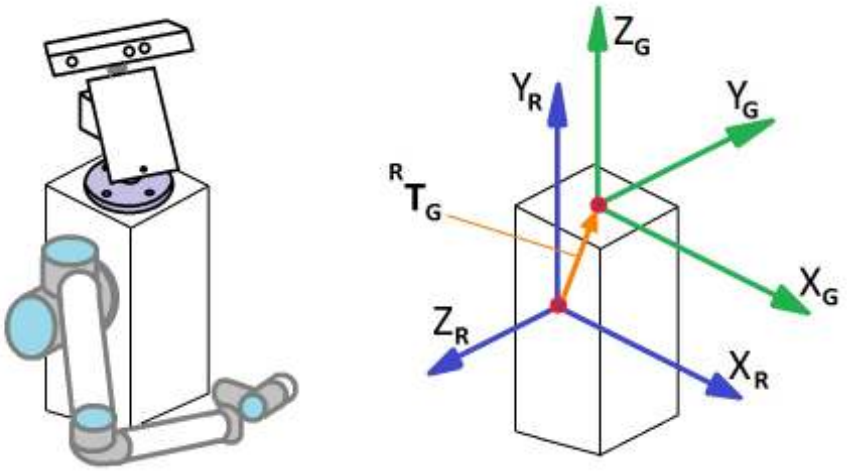
Točke putanje gibanja mogu se odrediti direktno koristeći ovaj princip a mogu se odrediti i ručnim navođenjem u sučelju, namještanjem konfiguracije robota po osima x,y,z i kutovima zakreta rx, ry, rz ili zakretanjem pojedinih zglobova dok se ne postigne željena konfiguracija.

6.4.2. API sučelje

Ovaj način programiranja robotske ruke predstavlja „offline“ način programiranja. Kod njega je potrebno spojiti dodatni uređaj, primjerice računalo, na upravljačku jedinicu robotskog sustava putem ethernet priključka. Potom se na računalu piše program u nekom programskom jeziku koristeći određenu sintaksu i naredbe koje su definirane od strane proizvođača a predviđene su za ovaj način programiranja. Program se izvršava na računalu a određene naredbe za izvršavanje nekih funkcija robota se šalju na upravljačku jedinicu robota putem TCP/IP protokola.

6.5. Kooperacija robotske ruke i robotske glave

Za ovu primjenu potrebno je povezati računalo i robotsku ruku putem ethernet priključka, tako da oni međusobno mogu razmjenjivati informacije. Na privjesku za učenje je napravljen program koji pomiče vrh alata robotske ruke po definiranoj putanji kroz nekoliko točaka. Koordinate točaka te putanje šalju se putem ethernet komunikacije potom na računalo, koje te podatke zaprima, obrađuje i prilagođava u prikladan oblik upravljačkih signala za pokretanje postolja. Nakon obrade tih signala računalo ih šalje na Arduino upravljačku jedinicu koja potom zakreće postolje, orijentirajući ga u smjeru vrha alata robota. Da bi se to ostvarilo, potrebno je još osim ostvarivanja ove komunikacije između robota i računala, odrediti i međusobnu ovisnost koordinatnih sustava robota i postolja pošto se ti sustavi ne nalaze u istoj točci a nisu ni jednako orijentirani u prostoru. Da bi se to ostvarilo potrebno je napraviti transformaciju koordinata robota u koordinatni sustav postolja.



Slika 31. Odnos koordinatnih sustava robotske ruke i robotske glave

Na slici iznad (**Slika 31**) lijevo, može se vidjeti shematski plan fizičke izvedbe sustava glave humanoidnog robota i robotske ruke UR5. Na desnoj strani slike vidi se položaj i orijentacija koordinatnih sustava: glave – označen zelenom bojom i pripadajućim osima X_G, Y_G i Z_G te robota – označen plavom bojom i prpadajućim osima X_R, Y_R i Z_R. Zbog tog pomaka između koordinatnih sustava a kako bi gibanje postolja glave bilo usklađeno sa gibanjem robotske ruke potrebno je odrediti njihovu međusobnu ovisnost. Ta ovisnost definira se određivanjem matrice transformacije koja se sastoji od matrice rotacije i matrice translacije, a predstavlja transformaciju iz koordinatnog sustava robota u koordinatni sustav robotske glave. Funkcija transformacije sa matricom ${}^{R}T_G$ označena je na slici iznad (**Slika 31**) narančastom bojom. Oba su koordinatna sustava kartezijska, pravokutni su i desnokretni.

Vrsta koordinatnog sustava i oblik postolja za montažu u obliku kvadra pojednostavljuje cijeli proces transformacije sustava jednog u drugi. Kako su sve plohe postolja međusobno okomite znači da ne postoji rotacija između sustava već samo imamo translaciju po osima a ona ovisi o dimenzijama postolja i orijentacijama osi.
Matematički zapisano, izrazi za transformaciju koordinata [12] iz sustava robotske ruke u sustav glave imaju slijedeći oblik:

$$P_G(X_G, Y_G, Z_G) - točka\ u\ koordinatnom\ sustavu\ robotske\ glave \quad \textbf{(4)}$$

$$P_R(X_R, Y_R, Z_R) - točka\ u\ koordinatnom\ sustavu\ robotske\ ruke \quad \textbf{(5)}$$

$$P_G = {}^{R}T_G \cdot P_R \quad \textbf{(6)}$$

$$\begin{bmatrix} X_G \\ Y_G \\ Z_G \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -H_P \\ 0 & 1 & 0 & -L_P \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} X_R \\ Y_R \\ Z_R \\ 1 \end{bmatrix} \quad \textbf{(7)}$$

Gdje su:

$H_P - udaljenostkoordinatnogsustavarobotaodgornjeplohepostolja$

$L_P - udaljenostkoordinatnogsustavaglaveodbočneplohepostolja$

7. PROGRAMSKA IZVEDBA

7.1. Kinect – glavna aplikacija [13]

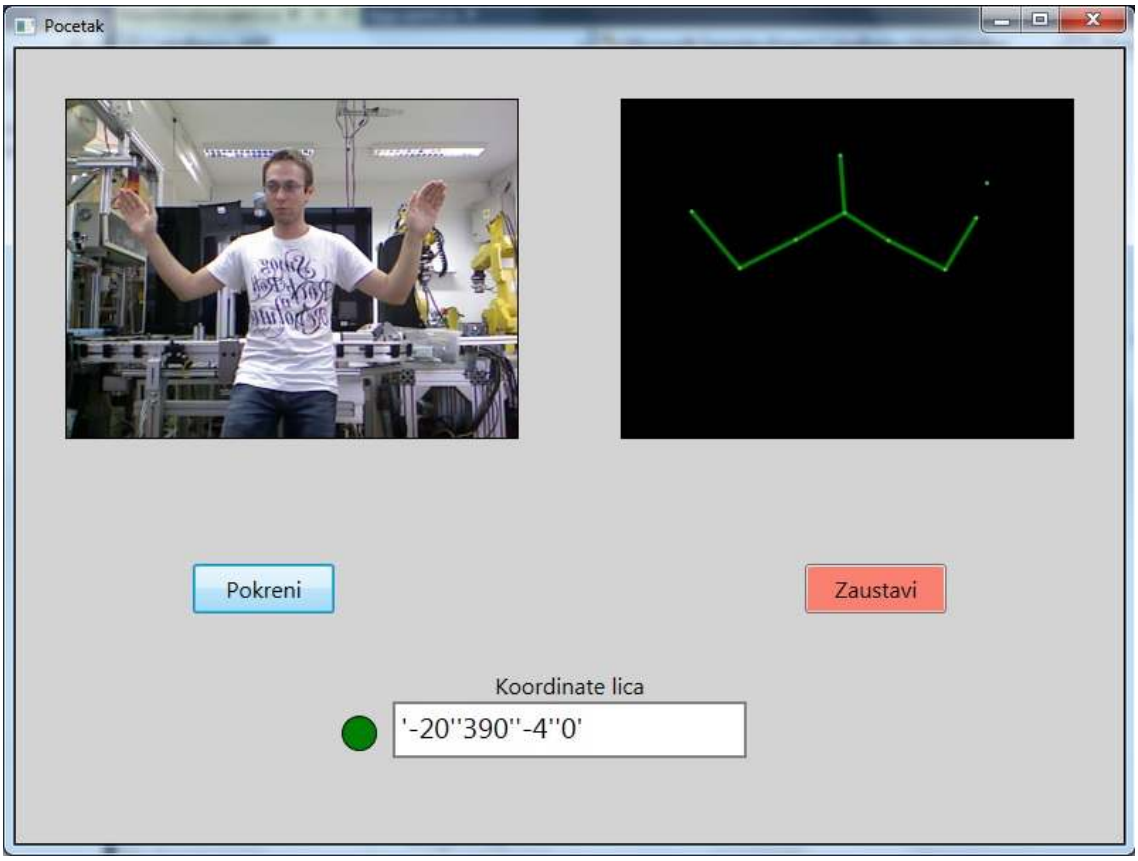
Razvoj aplikacija za korištenje funkcija Kinect-a omogućen je primjenom razvojnog okruženja Kinect for Windows SDK (od eng. Software Development Kit – razvojno programsko sučelje) i primjenom programskog sučelja Windows Visual Studio. Kako je navedeno u poglavlju koje opisuje razvoj Kinect-a i prateće programske podrške postoji četiri verzije razvojnog sučelja Kinect for Windows SDK a u izradi ovog rada korištena je poslijednja verzija sučelja, Kinect for Windows SDK 1.8. Za pisanje programskog dijela aplikacije korišteno je programsko sučelje Microsoft Visual Studio, verzija 2013, a programski jezik u kojem je aplikacija razvijena je C#. Za pomoć prilikom razvoja aplikacije u radu su korištena još Kinect-ova prateća sučelja Developer Toolkit Browser sa bazom podataka i raznih primjera aplikacija te Kinect Studio za provjeravanje i postavljanje osnovnih funkcija Kinect-a.

Aplikacija sa Kinectom predstavlja temeljni dio sustava robotizirane glave jer se na njega povezuju svi ostali dijelovi sustava koji komuniciraju međusobno putem aplikacije. Aplikacija je zamišljena kao program koji dobiva podatke sa Kinect-a i od robotske ruke te podatke obrađuje i dalje šalje prema sustavu upravljanja gibanja glave tj. prema Arduinu i ostalim elementima u upravljačkom lancu.

Temeljne funkcije koje aplikacija obavlja su slijedeće:

- Inicijalizacija i pokretanje Kinect-a
- Pokretanje kamera i slanja podataka
- Praćenje i prepoznavanje skeleta u vidnom polju Kinect-a
- Očitavanje položaja tj. koordinata glave korisnika u prostoru
- Izračun razlike relativnog položaja glave u odnosu na Kinect
- Prilagođavanje vrijednosti za upravljačke potrebe
- Komunikacija sa Arduino upravljačkim sustavom putem serijske veze (putem USB priključka)
- Komunikacija sa Universal Robotom UR5 putem ethernet veze

Da bi se rad cijelog sustava i iznosi vrijednosti određenih varijabli lakše nadzirao, u aplikaciji je integrirano grafičko sučelje koje objedinjuje osnovne elemente aplikacije i prikaz značajnijih vrijednosti pojedinih varijabli. Grafičko sučelje aplikacije je izrađeno u XAML programskom sučelju koje je integrirano u Microsoft Visual Studiu a omogućava prilično brzu i jednostavnu izradu grafičkih sučelja raznih namjene. XAML sučelje omogućava izradu grafičkih sučelja jednostavnim slaganjem elemenata poput tipki, klizača, padajućih izbornika, polja za ispis ili upis podataka, slikovne prikaze i tome sl. [17]



Slika 32. Izgled grafičkog sučelja aplikacije

Za potrebe aplikacije sa Kinect-om, u radu je izrađeno grafičko sučelje koje sadrži dva prozora za slikovni prikaz, lijevi prozor je za slike koje sa Kinect-a dolaze sa video kamere a desni prozor je za prikaz slika od funkcije praćenja skeleta korisnika u prostoru. Ovi prozori su izvedeni kao pojedinačni prikazi slika te se za potrebe dobivanja video prikaza sa Kinect-a učitavaju slijedno slike u određenom vremenskom razmaku, čime se dobije efekt video prikaza. Ovo slijedno očitavanje slika sa Kinect-a se naziva stream (eng. stream - tok) a slike se učitavaju brzinom prikaza od 30 slika u sekundi.

Oba prikaza olakšavaju korisniku pozicioniranje u vidnom prostoru Kinect-a jer je u slučaju bez prikaza dosta teško odrediti što Kinect „vidi“ a što ne. U sučelju se još nalaze dvije tipke, lijeva tipka zelene boje služi za pokretanje aplikacije i svih njenih funkcija dok desna tipka, crvene boje, služi za zaustavljanje rada aplikacije kao i svih funkcija koje ona obavlja. Nadalje, pri dnu glavnog prozora sučelja se nalazi prostor za ispis vrijednosti koordinata glave korisnika koji se prati u vidnom polju te mali kružić koji indicira stanje praćenja korisnika. U stanju dok se korisnik ne nalazi u vidnom polju Kinect-a ili zbog nekog drugog problema funkcija praćenja skeleta korisnika ne radi ovaj indikator je žute boje. Ukoliko funkcija praćenja skeleta korisnika radi i omogućeno je prepoznavanje i praćenje glave kao dio skeleta tada indikator poprima zelenu boju te signalizira aktivno praćenje glave korisnika.

U ovoj je primjeni funkcija praćenja skeleta prilagođena na način da algoritam prepoznavanja i praćenja skeleta prati samo gornji dio korisnikovog tijela: glavu, ramena i ruke, zbog vjerojatnosti da će se ispred robotizirane glave i robotske ruke nalaziti nekakav radni stol čime bi se onemogućilo algoritmu uobičajeno traženje i prepoznavanje donjeg dijela korisnikovog tijela, pa je upravo ovime to izbjegnuto.

Koordinate glave koje se dobiju primjenom funkcije praćenja skeleta (eng. Skeletal tracking) su u Kartezijskom koordinatnom sustavu (x, y, z) te ih je za potrebe upravljanja mehanizma potrebno prilagoditi. Ovo je dosta bitno jer je mehanizam glave izveden s dva rotacijska stupnja slobode gibanja te mu ove koordinate u Kartezijskom obliku bez dodatne prilagodbe i nisu od koristi.

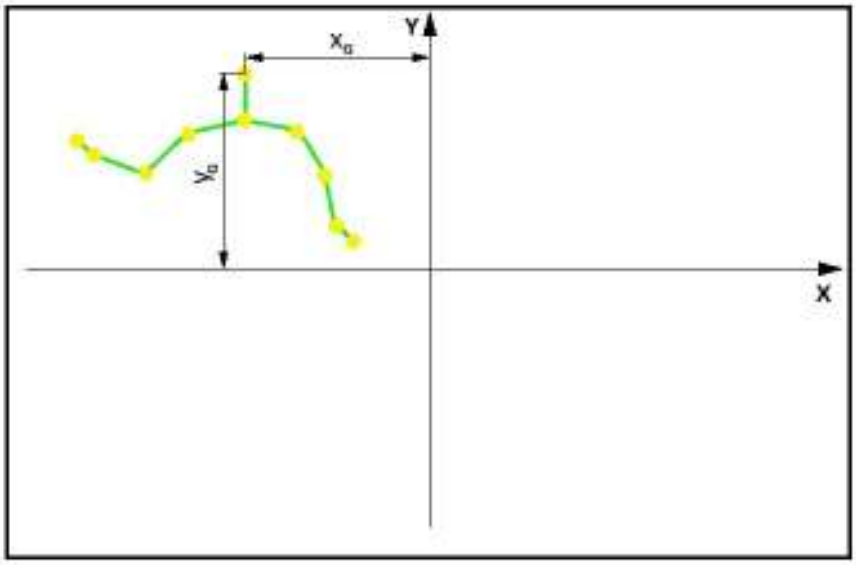
Zbog toga je sustav upravljanja gibanjem postolja izveden na način da se u aplikaciji očitavaju vrijednosti x i y koordinata glave korisnika te se potom uzima njihov iznos kao veličina za određivanje potrebnog pomaka pojedine osi. Vrijednost X koordinate glave korisnika se uzima kao referenca za upravljanje pomakom prve osi a vrijednost Y koordinate glave korisnika kao referenca za upravljanje pomakom druge osi mehanizma.

Ovdje su u programu još postavljeni i sljedeći izrazi:

$$\Delta X = X_t - X_{t-1} \tag{8}$$

$$v_x = \frac{\Delta X}{t} \tag{9}$$

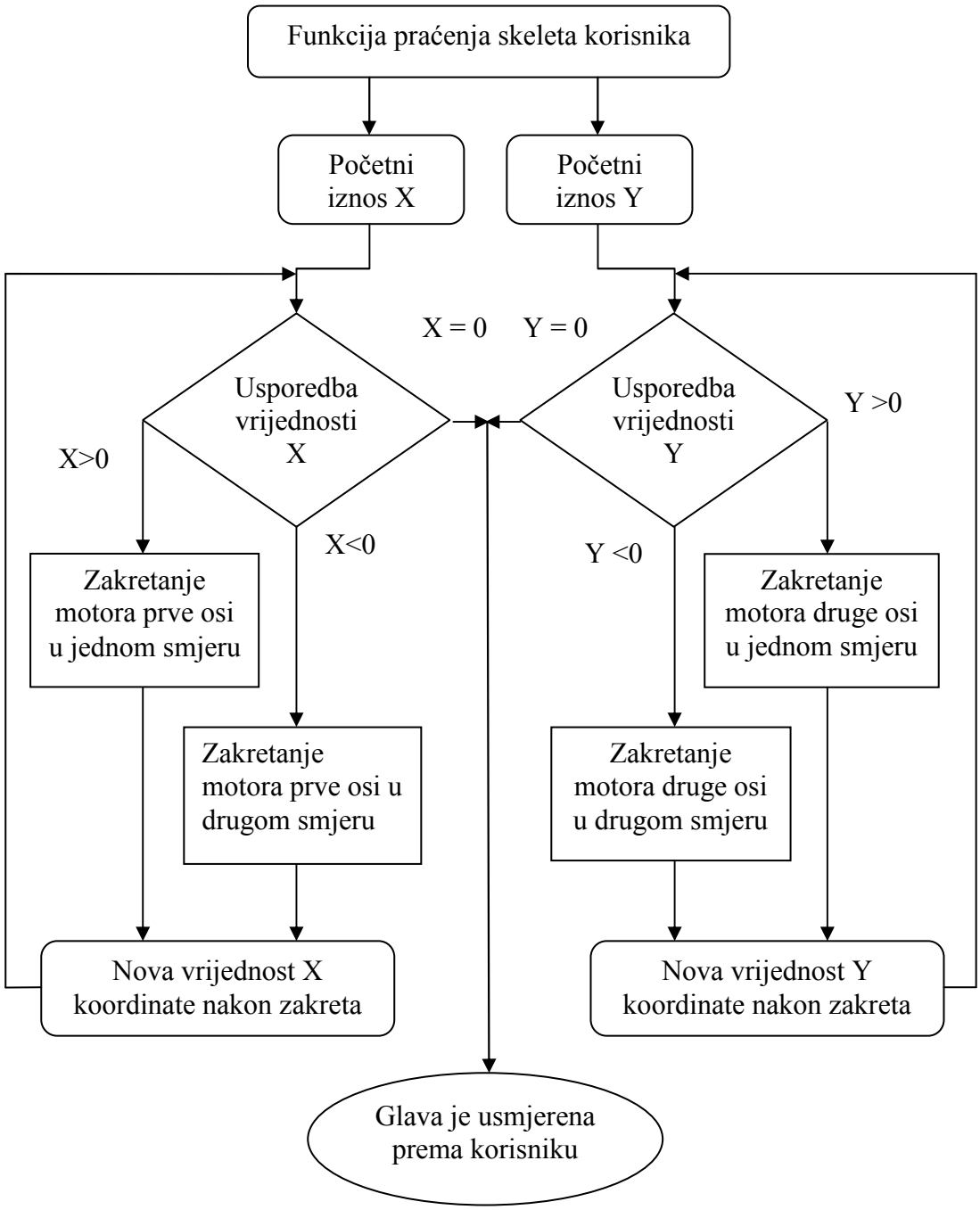
Iz prvog izraza se dobije vrijednost relativne promjene položaja točke u smjeru X osi u nekom vremenskom razdoblju a potom se primjenom drugog izraza poznavajući tu promjenu i trajanje vremenskog perioda između te dvije točke, može odrediti brzina pomaka točke u smjeru te osi. Analogno prethodno navedenome vrijedi i za drugu os. Ovo je korisna informacija ukoliko se gibanje sustava želi što više približiti stvarnom pomaku korisnika tj. kako bi dobili što realnije gibanje robotizirane glave. U ovom se slučaju ipak to odlučilo ne koristiti zbog nešto sporijeg prijenosnog omjera mehanizma robotizirane glave unatoč korištenju maksimalne brzine zakreta motora.



Slika 33. Praćenje glave korisnika u aplikaciji [15]

Vidljivo na prethodnoj slici (**Slika 33**), koordinate glave korisnika označene su sa X_G i Y_G. U ovisnosti o njihovim iznosima i predznaku određuju se parametri za pogon mehanizma. Svaki iznos koordinata glave koji je različit od nule označava da Kinect odnosno robotizirana glava, a na koju je on fiksno povezan, nije usmjerena prema korisniku te se automatski aktivira pogon mehanizma koji to želi ispraviti tj. usmjeriti Kinect skupa sa robotiziranom glavom u smjeru korisnika. Zbog velike preciznosti i osjetljivosti prilikom prepoznavanja i očitavanja koordinata glave korisnika nije moguće izvesti tako finu regulaciju zakreta mehanizma, zbog određenih odstupanja u prijenosnom mehanizmu, te su dopuštene vrijednosti koordinata postavljene na iznose nešto malo veće od nule. Ovime su se izbjegle

neželjene oscilacije i nestabilnost pogona u području oko ravnotežne točke. Dijagram toka upravljačkog lanca simbolički je prikazan u nastavku. (**Slika 34**)



Slika 34. Dijagram toka upravljačkog lanca

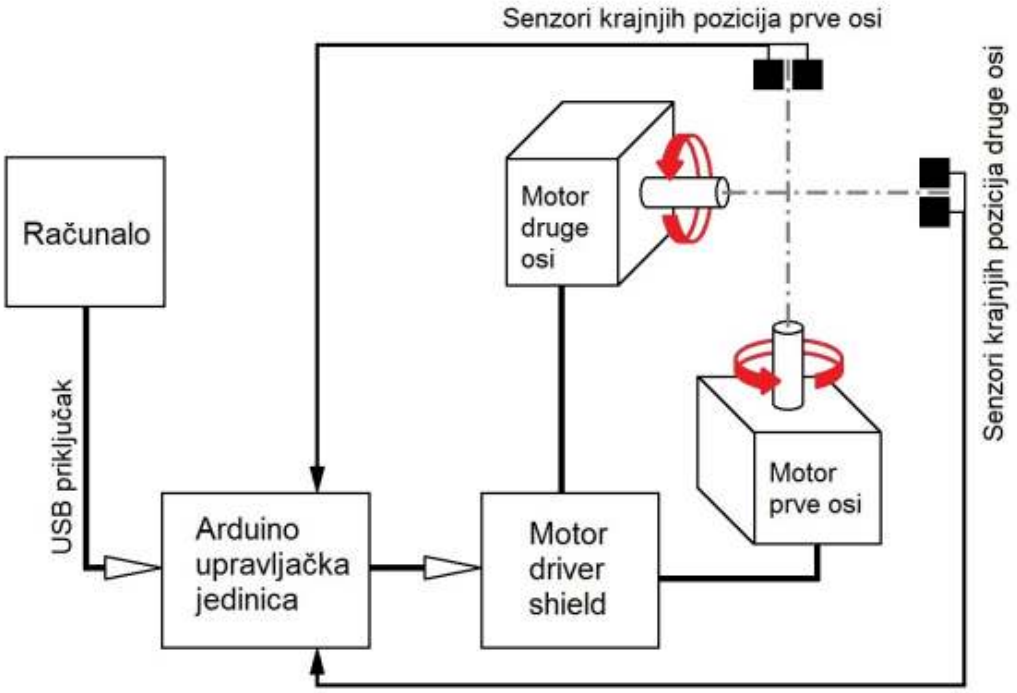
7.2. Arduino programski dio

Programski dio pisan u Arduino IDE (od eng. Integrated Development Environment-integrirano razvojno sučelje) sučelju a koje služi za komunikaciju s glavnom aplikacijom koja radi sa Kinectom i za upravljanje gibanjem mehanizma robotizirane glave. U glavnoj aplikaciji se izvršava program za procesiranje slike sa Kinect-a i izvršavanje funkcije prepoznavanja i praćenja skeleta korisnika. Iz funkcije praćenja skeleta se dobivaju te se kontinuirano prate i bilježe koordinate položaja korisnikove glave u radnom prostoru Kinecta.

Na osnovu tih koordinata se provodi daljnja prilagodba vrijednosti u oblik iskoristiv za upravljanje. U prethodnom poglavlju je detaljno opisan proces dobivanja koordinata iz funkcije praćenja skeleta dok će o procesu prilagodbe vrijednosti biti više riječi u slijedećem poglavlju.

7.2.1. Serijska komunikacija sa Kinect-om [16]

Arduino upravljačka jedinica je USB priključkom povezana na računalo na kojem se izvršava glavna aplikacija. Ovim priključkom je ostvarena serijska veza, putem koje se može iz glavne aplikacije na računalu pristupiti Arduino upravljačkoj jedinici tj. omogućena je njihova međusobna komunikacija.



Slika 35. Shema upravljačkog lanca robotske glave

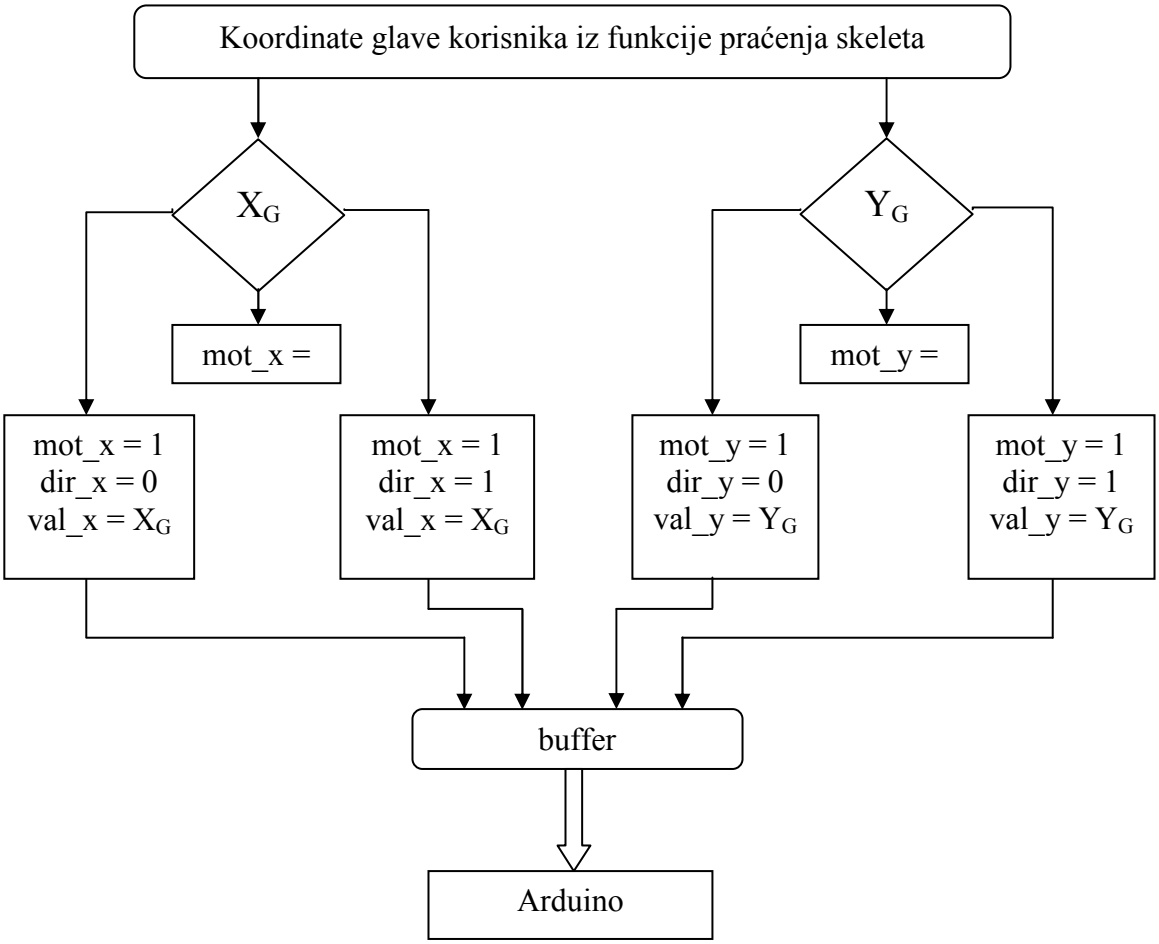
Kako je prethodno spomenuto koordinate dobivene funkcijom praćenja skeleta u glavnoj aplikaciji služe kao upravljačke veličine za pogon mehanizma robotizirane glave. Prilagođavanje njihovih vrijednosti za potrebe upravljanja obuhvaća ispitivanje te ograničavanje njihovih iznosa i ispitivanje njihovih predznaka. Za upravljanje se koristi 6 varijabli koje se mijenjaju u ovisnosti o iznosima navedenih koordinata. Varijable koje se koriste u programu su prikazane u slijedećoj tablici.

Tablica 8. Nazivi i opisi vrijednosti za upravljanje

Naziv	Vrijednost	Opis
mot_x	0	Označava isključivanje motora prve osi
	1	Označava uključivanje motora prve osi
mot_y	0	Označava isključivanje motora druge osi
	1	Označava uključivanje motora druge osi
dir_x	0	Označava smjer rotacije motora prve osi u jednu stranu
	1	Označava smjer rotacije motora prve osi u drugu stranu
dir_y	0	Označava smjer rotacije motora druge osi u jednu stranu
	1	Označava smjer rotacije motora druge osi u drugu stranu
val_x	>0	Označava iznos potrebnog zakreta prve osi
val_y	>0	Označava iznos potrebnog zakreta druge osi

Vrijednost varijable mot_x određuje se u programu na temelju vrijednosti x koordinate glave korisnika. Ukoliko se glava korisnika nalazi u ishodištu kordinatnog sustava Kinect-a po x osi tj. x koordinata glave korisnika iznosi nula, znači da robotiziranu glavu nije potrebno zakretati oko prve osi. Analogno tome vrijedi i za drugu os odnosno za vrijednost varijable mot_y. Nadalje, u ovisnosti o predznaku vrijednosti koordinata određuje se iznos varijabli dir_x i dir_y, ako je predznak negativan motor te osi se zakreće u jednu stranu tj. ako je predznak pozitivan motor se zakreće u drugu stranu. Na kraju, iznosi varijabli val_x i val_y se postavljaju kao apsolutne vrijednosti iznosa koordinata svake osi i predstavljaju vrijednost za koju je potrebno zakrenuti pojedinu os.

Sve dok je iznos neke od koordinata x ili y veći od nule, zakretat će se motori te osi te će se tako smanjivati odstupanje odnosno ishodište koordinatnog sustava Kinect-a će se uvijek nastojati pozicionirati identično sa položajem glave korisnika.



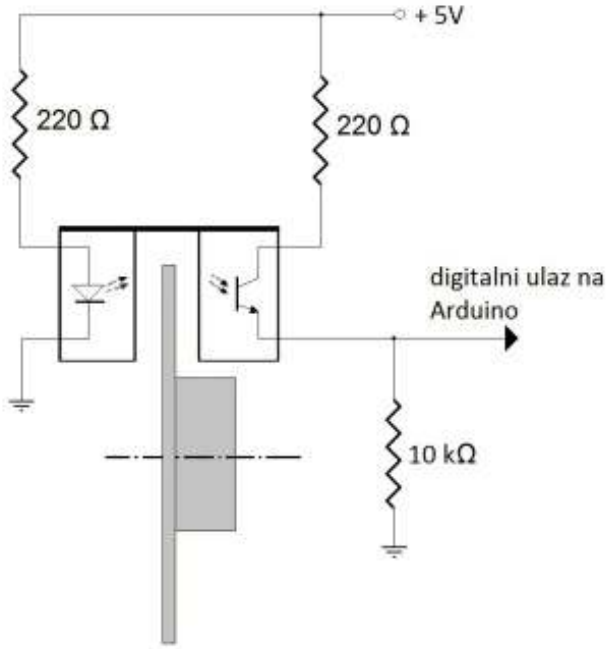
Slika 36. Postavljanje upravljačkih vrijednosti

Nakon procesa određivanja svih sešt vrijednosti potrebnih za upravljanje one se spremaju u tzv. memorijski buffer, koji ustvari predstavlja „paket“ podataka koji se šaljem nekom vezom prema drugim elementima sustava. U ovom slučaju buffer sadrži 7 podatkovnih mjesta, prvo mjesto u bufferu je brojčani podatak koji označava početak podatkovnog paketa koji se šalje vezom a preostalih 6 podataka su vrijednosti navedene u prethodnoj tablici (**Error! Reference source not found.**), potrebne za upravljanje.

Važno je samo točno odrediti prvi broj u bufferu i isti broj primjeniti u programskom kodu primatelja kako bi primatelj znao odrediti početak poruke. Nakon što programski kod primatelja prepozna početak poruke automatski se očitava šest vrijednosti nakon oznake početka poruke. Ovime je spriječeno očitavanje nekih neželjenih ili zalutalih vrijednosti i njihovo korištenje za upravljanje.

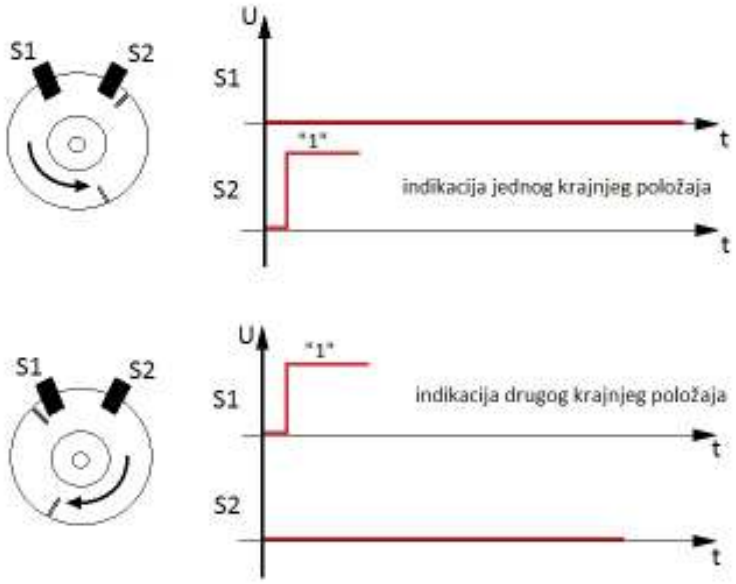
7.2.2. Inicijalizacija robotske glave

Mehanička izvedba sustava za inicijalizaciju položaja robotizirane glave je detaljno opisana u drugom poglavlju, a ovdje će biti opisan programski dio vezan za provedbu inicijalizacije. Sustav za inicijalizaciju, za svaku os, se sastoji od 2 optička senzora za indikaciju krajnjih pozicija i diska sa oznakama krajnjih položaja. Senzori su spojeni u električnom spoju tako da se koriste kao optičke „sklopke“ tj. spojene na Arduino njihovi se signali koriste se kao digitalni ulazi za indikaciju položaja.



Slika 37. Električni krug senzora krajnje pozicije

Električni spoj je identičan za oba položaja tako da za svaku os imamo spojena dva digitalna ulaza na Arduinu. Rad sustava inicijalizacije se temelji na principu optičke barijere koju predstavlja disk u normalnom položaju tj. u cijelom radnom području zakreta robotizirane glave. Krajnji položaji su označeni urezom na disku, čime se oslobađa optički put te senzor indicira nailazak na granicu radnog područja.



Slika 38. Indikacija krajnjih položaja

Programski je inicijalizacija izvedena tako da se prilikom pokretanja upravljačkog sustava oba motora, za svaku os, zakreću u jednu stranu sve dok sustav ne dođe do granice radnog područja. Nakon dostizanja te granice svaki se motor zakreće u suprotnu stranu za određeni broj koraka tako da dođe u sredinu radnog područja te ta točka predstavlja početni položaj sustava robotizirane glave.

Ukupni raspon radnog područja je poznat, za prvu os on iznosi 180° a za drugu 80°, te se preko prijenosnog omjera jednostavno izračuna potrebni broj koraka motora kako bi sustav došao na pola radnog područja te time izvršio postupak inicijalizacije. Potrebno je napomenuti da se ovaj postupak automatski izvršava samo prilikom pokretanja upravljačkog sustava a prema potrebi se može izvršavati još koji put i tijekom radnog procesa.

7.2.3. Program upravljanja motorima

Na Arduinu se izvršava programski kod za upravljački dio sustava robotizirane glave. Ovaj program radi neovisno o glavnoj aplikaciji sa Kinect-om, jedina je poveznica između njih serijska veza putem koje Arduino dobiva informacije potrebne za provedbu upravljanja radom mehanizma robotizirane glave. U programu su definirane sve vrijednosti parametara potrebnih za upravljanje i deklarirane su sve upravljačke varijable koje se koriste u programu upravljanja.

Program je strukturiran u nekoliko segmenata. Na početku je dio sa deklaracijom svih korištenih priključaka na Arduino upravljačkoj jedinici, deklarirane su sve varijable i sve ostale vrijednosti i konstante koje se koriste za upravljanje. Potom slijedi dio programa tzv. setup funkcija u kojoj se određuju vrste korištenih priključaka, da li je korišteni priključak ulaz ili izlaz signala, definiraju se postavke potrebne za komunikaciju kao što su vrsta veze i brzina prijenosa podataka i sl. Ovdje su definirani svi priključci svakog motora na 4x4 driver shield te redoslijed izvršavanja koraka po fazama motora.

Važno je napomenuti da se ovaj dio programa izvršava samo jednom i to prilikom prvog izvršavanja programskog koda, stoga se sve funkcije koje je potrebno izvršiti samo jednom, postavljaju u ovaj dio programskog koda. Kako je spomenuto u prethodnom dijelu, proces inicijalizacije sustava robotizirane glave je također potrebno izvršiti na početku tj. jednom prilikom pokretanja sustava. Funkcija koja vrši inicijalizaciju sustava se u ovom dijelu programskog koda izvršava pozivom te funkcije a temeljni algoritam i deklaracija funkcije se nalaze izvan ovog setup dijela programskog koda. Izdvojenim deklariranjem funkcije je programski kod bolje strukturiran, pregledniji je te ujedno smanjuje opterećenje mikrokontrolera prilikom izvršavanja programa.

Nakon dijela programa (setup funkcije) sa postavkama potrebnim za upravljanje slijedi dio korisnički deklariranih funkcija za izvršavanje raznih zadaća. Između ostalih, u ovom je dijelu definirana i funkcija za izvršavanje inicijalizacije sustava robotizirane glave.

Posljednji dio programskog koda je glavna petlja programa tzv. loop funkcija, u kojoj se programski kod unutar nje izvršava kontinuirano u petlji. Prekid izvršavanja ovog dijela programa je omogućen jedino ispunjavanjem postavljenih uvjeta u petlji ili nailaskom na naredbu za prekid izvršavanja programa. U ovom je dijelu programskog koda prema tome smješten dio programa koji služi za komunikaciju sa glavnom aplikacijom jer je to jedini dio sustava koji se mora kontinuirano izvršavati. U njemu je postavljena varijabla u koju se spremaju svi podaci pristigli serijskom vezom. Ova varijabla cijelo vrijeme očitava stanje serijske veze očekujući tako signal za početak čitanja i spremanja željenih korisnih podataka potrebnih za upravljanje.

U trenutku primanja podatka koji je definiran kao početak poruke (u glavnoj aplikaciji je to prvi podatak, od sedam, koji se sprema u memorijski buffer) tj. paketa podataka koji dolaze serijskom vezom označava se početak čitanja i spremanja svih šest vrijednosti, potrebnih za upravljanje. Svih šest vrijednosti se spremaju, uz vremenski razmak serijske

veze, u posebne varijable čija se stanja nadalje ispituju i prema njima se provodi upravljanje. Vrijednosti varijabli koje određuju potom daljnje potrebne radnje opisane su u poglavlju o serijskoj komunikaciji sa glavnom aplikacijom.(**Error! Reference source not found.**)

7.3. UR5 aplikacija [11]

Jedan dio sustava u radu je gibanje mehanizma robotizirane glave u kooperaciji sa radom robotske ruke UR5. Potrebno je povezati industrijsku robotsku ruku sa šest stupnjeva slobode gibanja u sustav robotizirane glave kako bi se ostvarilo gibanje mehanizma robotizirane glave u ovisnosti o gibanju robotske ruke. Zamisao je bilo napraviti sustav tako da robotska ruka UR5 obavlja nekakvu funkciju tj. da se giba u vidnom polju robotizirane glave a da se pri tome mehanizam glave usmjerava prema mjestu obavljanja nekakve zadaće a što je najčešće položaj poslijednjeg elementa robotske ruke odnosno TCP (eng. Tool Center Point – točka koja označava vrh alata robota).

Za početak je ostvarena veza robotske ruke i aplikacije za pogon mehanizma robotizirane glave međusobnim povezivanjem upravljačke jedinice robotske ruke na računalo na kojem se izvršava aplikacija upravljanja robotiziranom glavom. Komunikacija je ostvarena ethernet mrežom primjenom TCP/IP protokola.

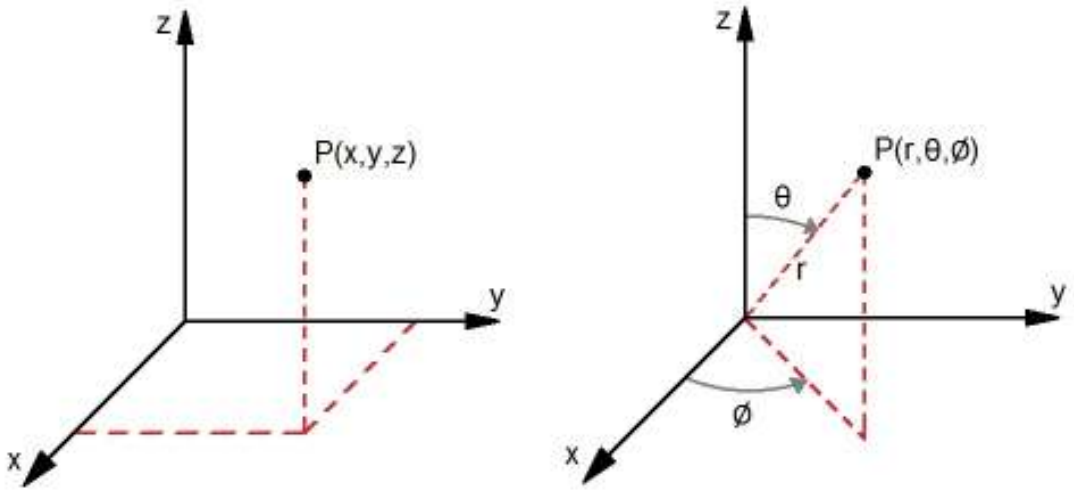
Kooperacija je izvedena tako da je u upravljačkom sustavu robotske ruke, pomoću privjeska za učenje, napisan program sa naredbama za obavljanje određenih radnji. U programu su uz naredbe za gibanje korištene i slijedeće funkcije:

- ***socket_open****(„IP adresa“, broj port-a)* - za otvaranje komunikacijskog kanala mrežnog priključka,
- ***get_target_tcp_pose****()* - za dohvaćanje podataka o trenutnom položaju TCP-a
- ***socket_send_string****(var)* - služi za komunikaciju putem ethernet mreže, točnije, za slanje varijable var u koju su pohranjeni podaci o položaju TCP-a

Aplikacija koja se izvršava na računalu dobiva podatke sa upravljačke jedinice robotske ruke i obrađuje ih te prilagođava za slanje prema upravljačkom dijelu robotizirane glave. Prilagodba vrijednosti koje aplikacija dobije obuhvaća sortiranje podataka jer podaci o položaju TCP-a dolaze kao string tj. kao slijed znakova a potrebno je imati zasebne informacije o x, y i z koordinati TCP-a. Kako su podaci u stringu posloženi redom kao

koordinate x,y i z te kutovi zakreta oko svake osi Rx, Ry i Rz a vrijednosti u stringu su razdvojene znakom zareza korištenjem funkcije za sortiranje stringa jednostavno se izuzmu potrebne prve tri vrijednosti iz stringa. Te se vrijednosti potom transformiraju matematičkim izrazima u Kartezijski koordinatni sustav robotizirane glave.

Matematički izrazi za ovu transformaciju su navedeni i obajšnjeni u petom poglavlju, u dijelu o kooperaciji robotske ruke i robotske glave. Iz Kartezijskog se sustava potom matematičkim izrazima ove koordinate transformiraju u sferni koordinatni sustav koji je vrlo sličan sustavu gibanja robotizirane glave.



Slika 39. Kartezijski (lijevo) i sferni (desno) koordinatni sustav

Koordinate točke u Kartezijskom koordinatnom sustavu robotizirane glave znamo te ih primjenom slijedećih matematičkih izraza transformiramo u sferni koordinatni sustav.

$$P(X_G, Y_G, Z_G) \rightarrow P(r, \theta, \emptyset) \tag{10}$$

$$r = \sqrt{{X_G}^2 + {Y_G}^2 + {Z_G}^2} \tag{11}$$

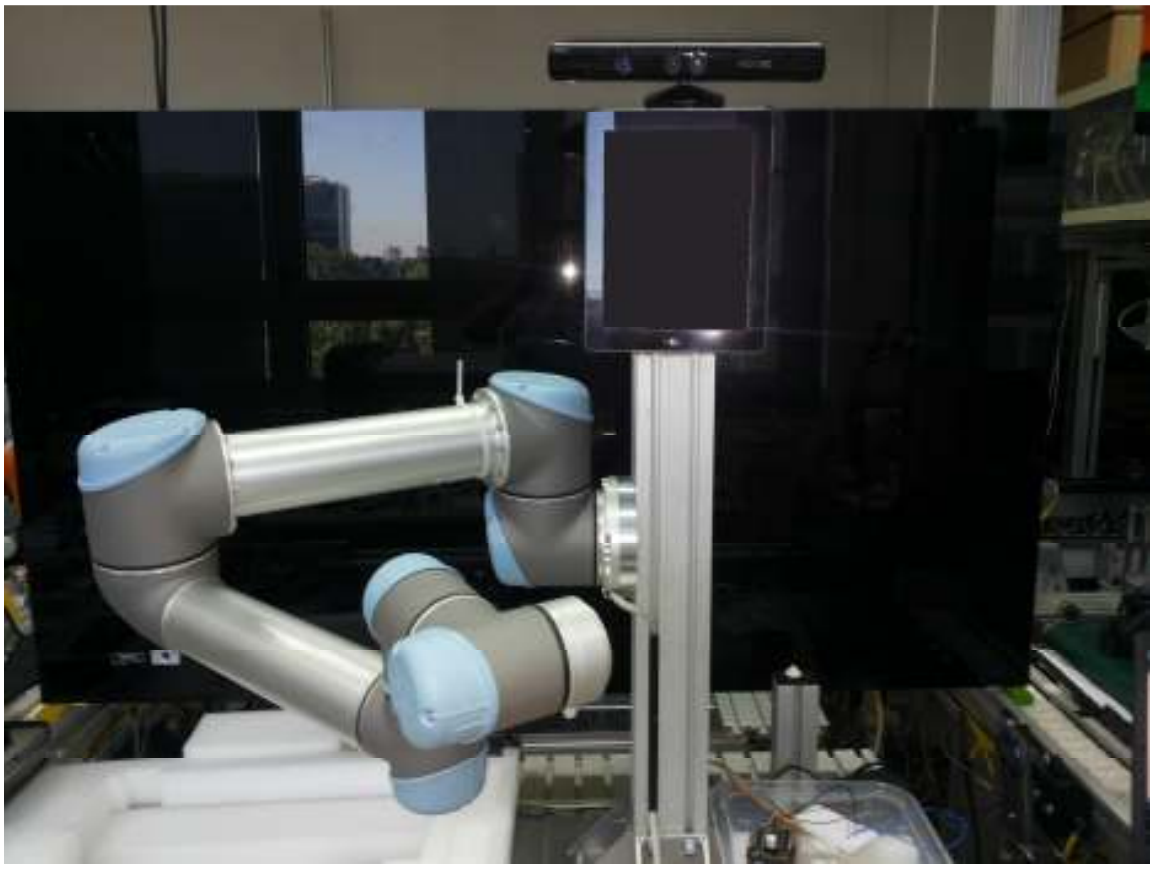
$$\theta = \arctan\left(\frac{\sqrt{{X_G}^2 + {Y_G}^2}}{Z_G}\right) \tag{12}$$

$$\emptyset = \arctan\left(\frac{Y_G}{X_G}\right) \tag{13}$$

Iz dobivenih koordinata u sfernom koordinatnom sustavu duljina radij vektora r nam nije od interesa jer nam udaljenost točke TCP-a od ishodišta koordinatnog sustava robotizirane glave ništa ne znači za upravljanje. Preostale dvije vrijednosti, θ i Ø, su kutni zakret radij vektora u odnosu na z os i kutni zakret projekcije radij vetora u x-y ravnini. Ove će se vrijednosti u aplikaciji koordiniranog rada robotizirane glave i robotske ruke koristiti za upravljanje radom motora robotizirane glave i to kut Ø za upravljanje prve osi a kut θ za upravljanje zakretom druge osi.

8. PRIKAZ FUNKCIJA RAZVIJENOG SUSTAVA

Nakon razrađene ideje cijelog sustava i nakon izrade robotske glave napravljeno je ožičenje dijela robotske glave. Fizička izvedba sustava automatskog upravljanja robotskom rukom i robotskom glavom je djelomično razrađena i opisana u prethodnim poglavljima u a u ovom će se poglavlju dati prikaz konačne izvedbe sustava sa svim njegovim elementima. Sustav je, shodno zadatku, zamišljen tako da se dobije što realističniji humanoidni karakter. Sastoji se od središnjeg postolja izvedenog od kvadratnog aluminijskog profila koji predstavlja „trup“ tj. osnovnu platformu na koju se dalje postavljaju preostali elementi. Ovaj je profil čvrsto povezan na temeljni stol na kojem je cijeli sustav izgrađen. Na postolje se s bočne strane, gledajući od naprijed, postavlja robotska ruka koja se koristi za zadatak kooperacije sa robotskom glavom. S gornje strane postolja se jednostavno postavlja sklop robotske glave koji je ključan dio za obavljanje zadataka automatskog upravljanja sustavom.



Slika 40. Postav sustava u laboratoriju

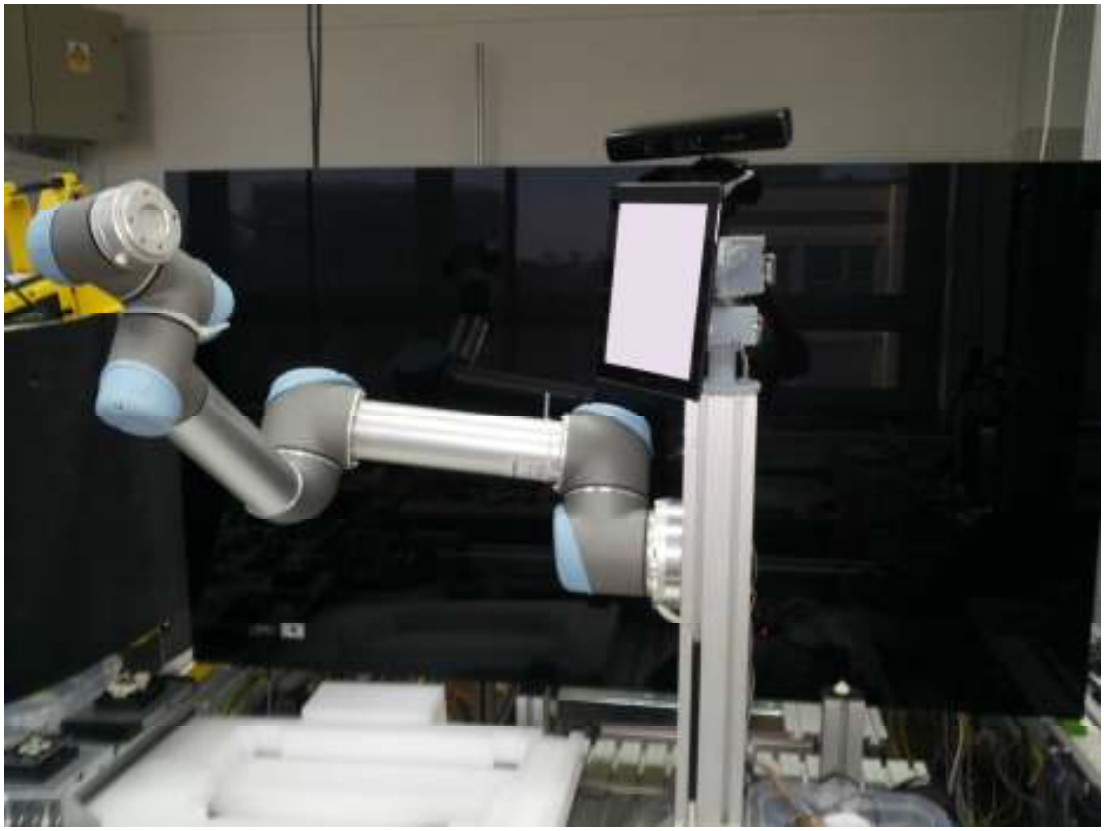
Iako je jedna od ideja za izvedbu cijelog sustava bila i korištenje dvije robotske ruke u sustavu, ovdje je sustav izveden prema trenutnim mogućnostima i radi jednostavnosti

primjenom samo jedne robotske ruke. Primjenom dviju robotskih ruku istovremeno dobio bi se bolji humanoidni karakter cijelog sustava.

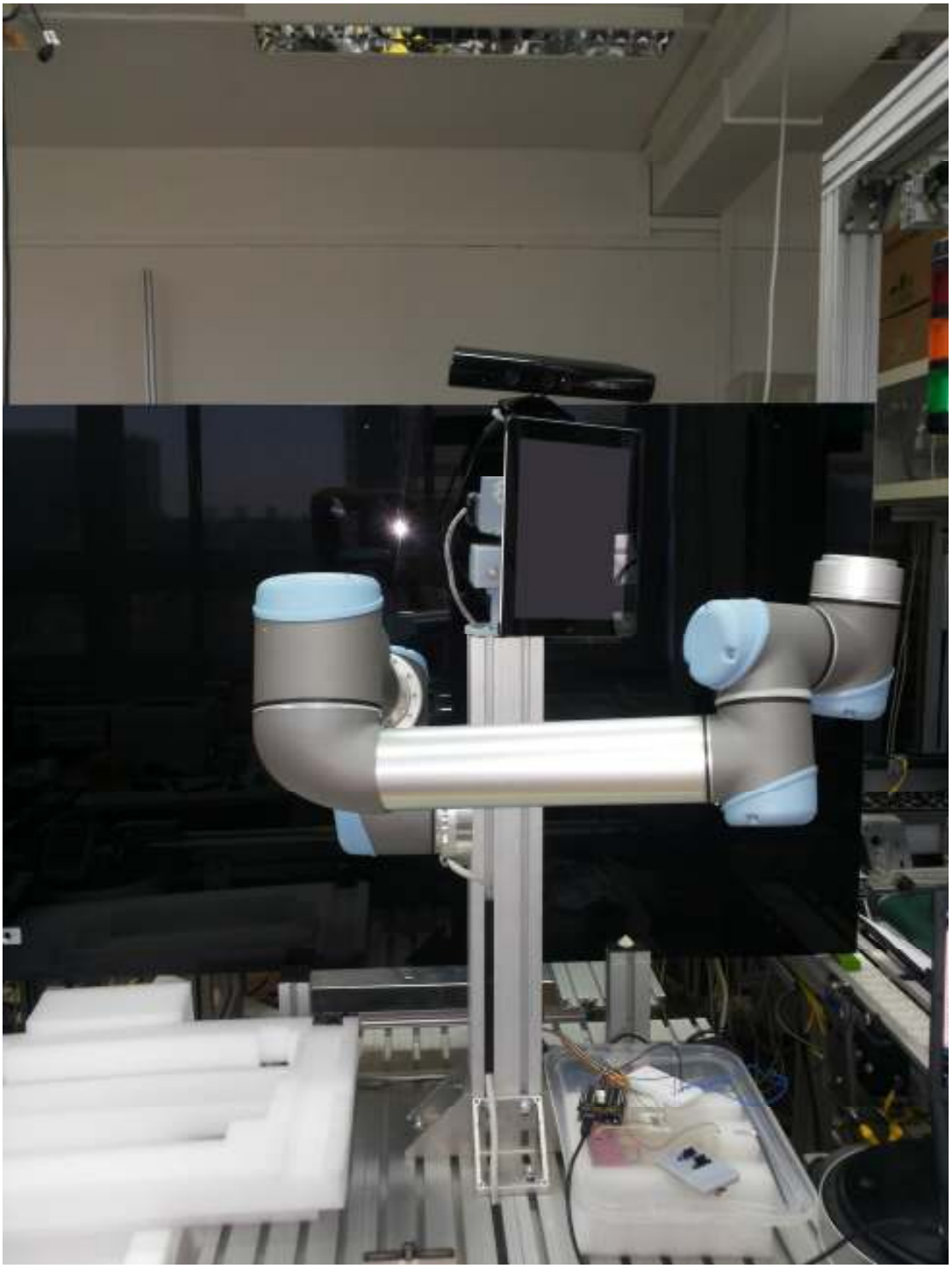
Nakon montaže svih dijelova sustava u jedan sklop, provedeno je testiranje obavljanja opisanih zadataka kooperacije robotske glave sa korisnikom te sa robotskom rukom. U zadatku kooperacije robotske glave sa korisnikom testirano je gibanje robotske glave zakretanjem po svakoj osi.

Pokazano je da se robotska glava može gibati u predviđenom radnom području u iznosu 180° prve osi (vertikalne osi) te u radnom području iznosa 60° kod druge (horizontalne) osi. Gibanje robotske glave se odvija uz nešto sporije gibanje od predviđenog zbog nedostataka prijenosnog mehanizma a koji su opisani u poglavlju koje opisuje sustav robotske glave sa svim dijelovima.

Drugi dio testiranja je proveden prema zadatku kooperacije robotske glave sa robotskom rukom. U zadatku je potrebno osigurati gibanje robotske glave u koordinaciji sa gibanjem robotske ruke tako da robotska glava prati vrh robotske ruke tj. alat i mjesto obavljanja neke operacije sa robotskom rukom.



Slika 41. Praćenje gibanja robotske ruke na desnu stranu



Slika 42. Praćenje gibanja robotske ruke u lijevu stranu

9. ZAKLJUČAK

U radu je izrađen i sklopljen mehanizam dvoosne robotske glave te je potom složen u sklop zajedno sa i industrijskom robotskom rukom. Upravljački sustav za gibanje robotske glave je također izrađen kao fleksibilna platforma za jednostavne izmjene ili nadogradnje sustava. U sklopu robotske glave primjenjeni su koračni motori za osiguranje pogona mehanizma glave te su također postavljeni i senzorski sklopovi koji reguliraju gibanje svakog motora u određenom radnom području. Primjena Kinect-a u sklopu robotske glave, programska podrška za praćenje skeleta koja je korištena u radu te ostali elementi ovog sustava omogućili su ostvarivanje zadaće praćenja korisnika prilikom kooperacije s njime.

Komunikacija je ostvarena između svih elemenata sustava i to ethernet vezom između robotske ruke i računala na kojem se izvršava glavna aplikacija te serijskom vezom između upravljačkog sustava robotske glave i računala.

U glavnoj aplikaciji, napisanoj u C# programskom jeziku, a koja se izvršava na računalu, integriran je dio sa korištenjem funkcija Kinect-a kao i dio za razmjenu podataka između elemenata sustava. Sustav automatskog upravljanja izvršava zadatke pronalaženja, prepoznavanja i praćenja korisnika u vidnom polju te usmjeravanje robotske glave u smjeru korisnika. Ovo se obavlja pomoću prije navedene programske aplikacije koristeći Kinect-ovu funkciju praćenja skeleta koja prepoznaje skelet korisnika te nam daje koordinate njegovog položaja u prostoru. Drugi zadatak koji sustav obavlja je kooperacija sa robotskom rukom pri kojoj upravljački sustav usmjerava robotsku glavu u smjeru točke vrha alata robotske ruke (TCP-a). Sa robotske ruke na računalo na kojem se izvršava glavni program, putem ethernet mreže se šalju podaci o položaju TCP-a robota te se u programu potom matematički prilagođavaju za upravljanje i šalju tako upravljačkom sustavu robotske glave. Na tabletu postavljenom na robotskoj glavi prikazuju se ekspresije lica te se time kompletnom sustavu daje humanoidni karakter.

Sustav automatskog upravljanja robotskom glavom predstavlja fleksibilnu platformu koja omogućava daljnju nadogradnju i ostvarivanje širokog spektra zadataka. Osim Kinect-ove funkcije praćenja skeleta, koja je ovdje korištena, moguće je u daljnjem razvoju iskoristiti i funkcije prepoznavanja govora kao i funkcija prepoznavanja gesti kako bi se ostvario još veći stupanj inteligencije sustava. Kao jedan od budućih zadataka bi se moglo napraviti, koristeći glasovne naredbe Kinect-a, glasovno upravljanje gibanja robotske glave.

10. LITERATURA

[1] Špičko, M.: Konstrukcija dvoosnog mehanizma robotske glave, 2015., Završni rad

[2] Decker, K. H.: Elementi strojeva, Tehnička knjiga Zagreb, 2006.

[3] http://www.eminebea.com/en/engineering_info/rotary/steppingmotor/hybridmotor/cat/001.shtml

[4] http://www.vishay.com/docs/81147/tcst2103.pdf

[5] https://www.arduino.cc/en/Main/arduinoBoardUno

[6] http://www.logos-electro.com/store/4x4-driver-shield

[7] http://www.orientalmotor.com/technology/articles/step-motor-basics.html

[8] https://en.wikipedia.org/wiki/Kinect

[9] Tölgyessy, M. , Hubinsky, P.: The Kinect sensor in robotics education, Institute of Control and Industrial Informatics, Slovak University of Technology in Bratislava

[10] http://www.universal-robots.com/en/

[11] http://www.wmv-robotics.de/home_htm_files/scriptmanual_en_1.5.pdf

[12] Bajd, T., Mihelj, M., Lenarčič, J. i dr.: Robotics, Springer, 2010.

[13] Kendal, S.: Object Oriented Programming using C#, BookBoon, 2011.

[14] Vidaković, J: Inteligentno vođenje robota pomoću stereo-vizijskog sustava, 2014., diplomski rad

[15] https://msdn.microsoft.com/en-us/library/hh855381

[16] http://stackoverflow.com/questions/20587910/c-sharp-serial-communication-with-arduino

[17] http://dotneteers.net/blogs/vbandi/archive/2013/03/25/kinect-interactions-with-wpf-part-i-getting-started.aspx

11. PRILOZI

11.1. Programski kod

11.1.1. Kinect glavna aplikacija (C#)

```
namespace Microsoft.Samples.Kinect.ColorBasics
{
    using System;
    using System.Linq;
    using System.Threading;
    using System.Globalization;
    using System.IO;
    using System.IO.Ports;
    using System.Net.Sockets;
    using System.Windows;
    using System.Windows.Controls;
    using System.Windows.Threading;
    using System.Windows.Input;
    using System.Windows.Media;
    using System.Windows.Media.Imaging;
    using Microsoft.Kinect;

    /// Interaction logic for MainWindow.xaml
    public partial class MainWindow : Window
    {

        // Active Kinect sensor
        private KinectSensor sensor;

        //definiranje SerialPorta za Arduino USB (oznaka priključka,
        baudrate)
        SerialPort arduino = new SerialPort("COM3", 9600);

       // postavljanje TCP priključka UR5 robotske ruke
       Tcplistener server = null;

       try
       {
            // Set the TcpListener on port 200
            Int32 port = 200;
            IPAddress localAddr = IPAddress.Parse("192.168.123.200");

            //TcpListener server = new TcpListener(port);
            server = new TcpListener(localAddr, port);

            // Start listening for client requests
            server.Start();
```

```
// Buffer for reading data
Byte[] bytes = new Byte[64];

// Definiranje upravljačkih varijabli za pogon robotske glave
float lastFi = 0;
float lastTh = 0;

bool URstate = false;

// Petlja očitavanja stanja pozicije TCP-a robotske ruke
while (true)
     {
         // Perform a blocking call to accept requests.
         TcpClient client = server.AcceptTcpClient();

         // Get a stream object for reading and writing
         NetworkStream stream = client.GetStream();

         int i;
         int br = 1;
         int Hp = 65;    // širina profila postava
         int Lp = 200;   // visina razmaka sustava postava

         while ((i = stream.Read(bytes, 0, bytes.Length)) != 0)
         {
             URstate = true;
             string s = Encoding.ASCII.GetString(bytes, 0, i);

             string st = s.Trim(new Char[] { ' ', 'p', '[', ']' });
             IList<string> koord = st.Split(new string[] { ",", " "
             },
             StringSplitOptions.RemoveEmptyEntries);

     // Ispis varijabli za kontrolu iznosa
          //   Console.WriteLine("Tocka {0}", br);
          //   Console.WriteLine("Koordinata X: {0}", koord[0]);
          //   Console.WriteLine("Koordinata Y: {0}", koord[1]);
          //   Console.WriteLine("Koordinata Z: {0}", koord[2]);
          //   Console.WriteLine("-----------------------------");
          //   br++;

     // Izracunavanje upravljackih vrijednosti za pogon robotske
        glave u koordinaciji sa robotskom rukom UR5
     // koordinate robotske ruke
     XR = koord[0];
     YR = koord[1];
     ZR = koord[2];
```

```
// transformacija koordinata u prostor robotske glave
XG = XR;
YG = -ZR-Hp;
ZG = YR-Lp;

Fi = atan(YG/XG);
Th = atan(Math.Sqrt(Math.Pow(XG, 2) + Math.Pow(YG, 2)) / ZG);

// dobivene vrijednosti kutova su u radijanima pa ih za
upravljanje treba preko prijenosnog omjera mehanizma
prilagoditi broju koraka za potrebni zakret

Fi_step = (Math.Round(Fi)*6000) / (2*Math.PI);
Th_step = (Math.Round(Th)*6000) / (2*Math.PI);

if(Fi != lastFi)
{
     UR_mot_x = 1;
     UR_val_x = Fi_step;
}
else
{
     mot_x = 0;
}
lastFi = Fi;

if(Th != lastTh)
{
     UR_mot_y = 1;
     UR_val_y = Th_step;
}
else
{
     mot_y = 0;
}


lastTh = Th;

    }
    // Prekid veze s robotom
    client.Close();
    }
}
catch (SocketException e)
{
    //Console.WriteLine("SocketException: {0}", e);
}
finally
{
    // Stop listening for new clients.
```

```
            server.Stop();
            //Console.WriteLine("Server stop");
        }
      }
    }

     //Određivanje širine slike prikaza(piksela)
     private const float RenderWidth = 640.0f;

     //Određivanje visine slike prikaza(piksela
     private const float RenderHeight = 480.0f;

     //Debljina crta za prikaz skeleta korisnika
     private const double JointThickness = 3;

     //Debljina crta elipse za prikaz centra tijela
     private const double BodyCenterThickness = 10;

     //Thickness of clip edge rectangles
     private const double ClipBoundsThickness = 10;

     //Alat za iscrtavanje središnje točke skeleta
     private readonly Brush centerPointBrush = Brushes.Blue;

     //Alat za iscrtavanje crta između poznatih zglobova
    private readonly Brush trackedJointBrush = new SolidColorBrush
   (Color.FromArgb(255, 68, 192, 68));

     //Alat za iscrtavanje zglobova koji se preklapaju
     private readonly Brush inferredJointBrush = Brushes.Yellow;

     //Alat za iscrtavanje praćenih kostiju skeleta
     private readonly Pen trackedBonePen = new Pen(Brushes.Green, 6);

     // Alat za iscrtavanje kostiju koje se preklapaju
     private readonly Pen inferredBonePen = new Pen(Brushes.Gray, 1);

     //Grupa za izlazni render skeleta
     private DrawingGroup drawingGroup;

     //Slika koja ce se prikazati na zaslonu
     private DrawingImage imageSource;

     // Bitmap datoteka koja sadržava podatke o bojama u prikazu
     private WriteableBitmap colorBitmap;

     // Intermediate storage for the color data received from the camera
     private byte[] colorPixels;

     double Gxo, Gyo, Xglave, Yglave, X_data, Y_data;
     int Xint_data, Yint_data;
```

```
double v_x, v_y, vrat,Yramena, vratLen, v;
bool glavaId;

double[] Xgl = new double[5000];
double[] Ygl = new double[5000];

// Varijable za izbor motora i smjera rotacije
int count, br;
byte mot_x, mot_y, dir_x, dir_y;

// Varijable za izbor brzine rotacije i iznos cilja
int val_x, val_y, brz_x, brz_y;

// Initializes a new instance of the MainWindow class
public MainWindow()
{
    InitializeComponent();
}

/// Execute startup tasks ____________________________________________
private void WindowLoaded(object sender, RoutedEventArgs e)
{
    Gxo = 0;
    Gyo = 0;
    count = 0;
    br = 0;

    //Create the drawing group we'll use for drawing
    this.drawingGroup = new DrawingGroup();

    //Create an image source that we can use in our image control
    this.imageSource = new DrawingImage(this.drawingGroup);

    //Display the drawing using our image control
    Slika_glava.Source = this.imageSource;

    // Look through all sensors and start the first connected one.
    // This requires that a Kinect is connected at the time of app
    // startup.
    // To make your app robust against plug/unplug, it is
    // recommended to use KinectSensorChooser provided in
    // Microsoft.Kinect.Toolkit (See components in Toolkit
    // Browser).

    foreach (var potentialSensor in KinectSensor.KinectSensors)
    {
        if (potentialSensor.Status == KinectStatus.Connected)
        {
            this.sensor = potentialSensor;
            break;
        }
    }
```

```
    if (null != this.sensor)
    {
    // Turn on the color stream to receive color frames
  this.sensor.ColorStream.Enable
  (ColorImageFormat.RgbResolution640x480Fps30);

//Turning on depth stream
this.sensor.DepthStream.Enable(DepthImageFormat.Resolution640x480Fp
s30);

// Allocate space to put the pixels we'll receive
this.colorPixels = new byte
[this.sensor.ColorStream.FramePixelDataLength];

// This is the bitmap we'll display on-screen
    this.colorBitmap = new WriteableBitmap
    (this.sensor.ColorStream.FrameWidth,
    this.sensor.ColorStream.FrameHeight, 96.0, 96.0,
    PixelFormats.Bgr32, null);

// Set the image we display to point to the bitmap where we'll put
   the image data
this.Slika_RGB.Source = this.colorBitmap;

// Add an event handler to be called whenever there is new color
   frame data
this.sensor.ColorFrameReady += this.SensorColorFrameReady;

// Add an event handler to be called whenever there is new depth
   frame data ready
this.sensor.DepthFrameReady += this.SensorDepthFrameReady;

// Turn on the skeleton stream to receive skeleton frames
this.sensor.SkeletonStream.Enable();

// Setting up skeleton tracking mode: default(standing) or seated
this.sensor.SkeletonStream.TrackingMode =
SkeletonTrackingMode.Seated;

// Add an event handler to be called whenever there is new color
   frame data
   this.sensor.SkeletonFrameReady +=
   this.SensorSkeletonFrameReady;
   }
    if (null == this.sensor)
    {
        koord.Text = "senzor nije spreman";
    }
    DispatcherTimer dispatcherTimer = new DispatcherTimer();
    dispatcherTimer.Tick += new EventHandler(dispatcherTimer_Tick);
    dispatcherTimer.Interval = new TimeSpan(0, 0, 0, 0, 50);
```

```
            dispatcherTimer.Start();
        }

        /// Execute shutdown tasks
       private void WindowClosing(object sender,
                             System.ComponentModel.CancelEvent   Args e)
        {
            if (null != this.sensor)
            {
                this.sensor.Stop();
            }
        }
        /// Event handler for Kinect sensor's ColorFrameReady event
       private void SensorColorFrameReady(object sender,
      ColorImageFrameReadyEvent Args e)
        {
            using (ColorImageFrame colorFrame = e.OpenColorImageFrame())
            {
                if (colorFrame != null)
                {
                    // Copy the pixel data from the image to a temporary
                       array
                    colorFrame.CopyPixelDataTo(this.colorPixels);

                    // Write the pixel data into our bitmap
                  this.colorBitmap.WritePixels(new Int32Rect(0, 0,
                  this.colorBitmap.PixelWidth,
                  this.colorBitmap.PixelHeight),
                  this.colorPixels,
                  this.colorBitmap.PixelWidth * sizeof(int), 0);
                }
            }
        }
        // Event handler za SensorSkeletonFrameReady_____________________
      private void SensorSkeletonFrameReady(object sender,
      SkeletonFrameReadyEvent Args e)
        {
            count++;
            if (count == 30)
            {
                count = 0;
            }
            Skeleton[] skeletons = new Skeleton[0];

            using (SkeletonFrame skeletonFrame = e.OpenSkeletonFrame())
            {
                if(skeletonFrame != null)
                {
                    skeletons = new
                    Skeleton[skeletonFrame.SkeletonArrayLength];
                    skeletonFrame.CopySkeletonDataTo(skeletons);
                }
```

```
            }

            using (DrawingContext dc = this.drawingGroup.Open())
            {
            dc.DrawRectangle(Brushes.Black, null, new Rect(0.0, 0.0,
            RenderWidth, RenderHeight));

                if(skeletons.Length != 0)
                {
                    foreach (Skeleton skel in skeletons)
                    {
                        if(skel.TrackingState ==
                        SkeletonTrackingState.Tracked)
                        {
                            this.DrawBonesAndJoints(skel, dc);
                        }
                    }
                }
                //Prevent drawing outside of our render area
            this.drawingGroup.ClipGeometry = new RectangleGeometry(new
            Rect(0.0, 0.0, RenderWidth, RenderHeight));
            }
        }
        // Event handler za DrawBonesAndJoints_______________________________
        private void DrawBonesAndJoints(Skeleton skeleton, DrawingContext
drawingContext)
        {
            // Torzo
            this.DrawBone(skeleton, drawingContext, JointType.Head,
            JointType.ShoulderCenter);

          this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter,
          JointType.ShoulderRight);

          this.DrawBone(skeleton, drawingContext, JointType.ShoulderCenter,
          JointType.ShoulderLeft);

            // Desna ruka
           this.DrawBone(skeleton, drawingContext, JointType.ShoulderRight,
           JointType.ElbowRight);

           this.DrawBone(skeleton, drawingContext, JointType.ElbowRight,
           JointType.WristRight);

           this.DrawBone(skeleton, drawingContext, JointType.WristRight,
           JointType.HandRight);

           // Lijeva ruka
            this.DrawBone(skeleton, drawingContext, JointType.ShoulderLeft,
            JointType.ElbowLeft);
```

```
this.DrawBone(skeleton, drawingContext, JointType.ElbowLeft,
JointType.WristLeft);

this.DrawBone(skeleton, drawingContext, JointType.WristLeft,
JointType.HandLeft);

// Render Joints
foreach(Joint joint in skeleton.Joints)
{
    Brush drawBrush = null;
    else if (joint.TrackingState == JointTrackingState.Tracked)
    {
        drawBrush = this.trackedJointBrush;
    }
    else if (joint.TrackingState ==
    JointTrackingState.Inferred)
    {
        drawBrush = this.inferredJointBrush;
    }
    if (drawBrush != null)
    {
        drawingContext.DrawEllipse(drawBrush, null,
      this.SkeletonPointToScreen(joint.Position),
       JointThickness,
      JointThickness);
    }
}
// Dio za spremanje koordinata glave i prikaza vrijednosti u
   prozoru aplikacije

// Definiranje glave kao zgloba koji ce se pratit u aplikaciji
Joint glava = skeleton.Joints[JointType.Head];
Joint ramena = skeleton.Joints[JointType.ShoulderCenter];

if (glava.TrackingState == JointTrackingState.Tracked)
{
    glavaId = true;
    signal_data.Fill = new SolidColorBrush
    (System.Windows.Media.Colors.Green);

    if(count == 0 || count == 6 || count == 12 || count == 18
       || count ==  24)
    {
        br++;
        // Koordinate glave u vidnom polju Kinecta
        Xgl[br] = Math.Round(glava.Position.X * 1000, 1);
        Ygl[br] = Math.Round(glava.Position.Y * 1000, 1);

        // Brzine gibanja tocke glave u X i Y smjeru
        v_x = (Xgl[br] - Xgl[br - 1]) / 0.2;
        v_y = (Ygl[br] - Ygl[br - 1]) / 0.2;
```

```
}

// Koordinate glave u ravnini
Xglave = Math.Round(glava.Position.X * 1000, 1);
Yglave = Math.Round(glava.Position.Y * 1000, 1);

// Iznos pomaka glave u koordinatnom sustavu XY ravnine
X_data = Math.Round((Xglave - Gxo) / 1, 0);
Y_data = Math.Round((Yglave - Gyo) / 1, 0);

// Ako se glava prati onda se ispisuju vrijednosti
   koordinata u grafičkom sučelju
if(glavaId == true)
{
    koord.Text = "'" + X_data + "'" + "'" + Y_data + "'" +
    "'" + v_x + "'" + "'" + v_y + "'";
}

Xint_data = Convert.ToInt16(X_data);
Yint_data = Convert.ToUInt16(Y_data);

val_x = Math.Abs(Xint_data);
val_y = Math.Abs(Yint_data);

if (val_x > 255)
{
    val_x = 255;
}
if (ramena.TrackingState == JointTrackingState.Tracked)
{
    Yramena = ramena.Position.Y;
    vrat = Math.Abs(glava.Position.Y - Yramena);
    vratLen = Math.Round(vrat * 1000, 0);
}
// Odredivanje motora i smjera rotacije motora za X os
// Ako je pozicija razlicita od nule tada se motor pogoni
if (X_data != 0)
{
    mot_x = 1;        // stanje motora ukljuci
    brz_x = 1;
    // ispitivanje pomaka glave u pozitivnom smjeru osi
    if(X_data > 0)
    {
    // postavljanje smjera rotacije u pozitivnom smjeru
       dir_x = 1;
    }

 // pomak glave u negativnom smjeru osi
    else
    {
        // postavljanje smjera rotacije u negativnom smjeru
        dir_x = 0;
```

```
                }
            }
        // Inace ako je pozicija jednaka nuli tada motor treba mirovati
            else
            {
                mot_x = 0;        // stanje motora iskljuci
            }
        // Odredivanje motora i smjera rotacije motora za Y os
        // Ako je pozicija razlicita od nule tada se motor pogoni
            if (Y_data != 0)
            {
                mot_y = 1;        // stanje motora ukljuci
                brz_y = 1;

                // ispitivanje pomaka glave u pozitivnom smjeru osi
                if(Y_data > 0)
                {
                    // postavljanje smjera rotacije u pozitivnom smjeru
                    dir_y = 1;
                }
                // pomak glave u negativnom smjeru osi
                else
                {
                    // postavljanje smjera rotacije u negativnom smjeru
                    dir_y = 0;
                }
            }

          // Inace ako je pozicija jednaka nuli tada motor treba mirovati
            else
            {
                mot_y = 0;        // stanje motora iskljuci
            }
        }
// Ako se glava ne prati onda se indikator u grafičkom sučelju
iscrtava u žutoj boji
else
        {
            glavaId = false;
            signal_data.Fill = new SolidColorBrush
            (System.Windows.Media.Colors.Yellow);
        }
 }

  private void dispatcherTimer_Tick(object sender, EventArgs e)
  {
 // Ako se prati glava korisnika
      if (glavaId == true)
      {
          arduino.Open();
```

```
            if (arduino.IsOpen)
            {
                byte[] buffer = new byte[7];

                // Pocetak poruke pocinje sa brojem 16 za uskladivanje
                // pocetka slanja i primanja paketa podataka sa Kinecta
                // na robotsku glavu

                buffer[0] = Convert.ToByte(16);

           // varijabla stanja pogona motora za os X
                buffer[1] = Convert.ToByte(mot_x);

           // varijabla cilja zakreta motora za os X
                buffer[2] = Convert.ToByte(val_x);

           // varijabla smjera zakreta motora za os X
                buffer[3] = Convert.ToByte(dir_x);

           // varijabla stanja pogona motora za os Y
                buffer[4] = Convert.ToByte(mot_y);

           // varijabla cilja zakreta motora za os Y
                buffer[5] = Convert.ToByte(val_y);

           // varijabla smjera zakreta motora za os Y
                buffer[6] = Convert.ToByte(dir_y);

                try
                {
                    arduino.Write(buffer, 0, 7);
                    Thread.Sleep(500);
                    arduino.Close();
                }
                catch
                {
                    Thread.Sleep(1);
                }
            }
         }
        else if (glavaId == false && URstate == true)
        {
           arduino.Open();

            if (arduino.IsOpen)
            {
                byte[] buffer = new byte[7];

                // Pocetak poruke pocinje sa brojem 14 za uskladivanje
                // pocetka slanja i primanja paketa podataka sa UR
                // robotske ruke na robotsku glavu.
```

```
                buffer[0] = Convert.ToByte(14);

            // varijabla stanja pogona motora za os X
                buffer[1] = Convert.ToByte(UR_mot_x);

            // varijabla iznosa kuta zakreta motora za os X
                buffer[2] = Convert.ToByte(UR_val_x);

            // varijabla stanja pogona motora za os Y
                buffer[3] = Convert.ToByte(UR_mot_y);

            // varijabla iznosa kuta zakreta motora za os Y
                buffer[4] = Convert.ToByte(UR_val_y);

            // Podatak koji se ne koristi
                buffer[5] = Convert.ToByte(0);

            // Podatak koje se ne koristi
                buffer[6] = Convert.ToByte(0);

                try
                {
                    arduino.Write(buffer, 0, 7);
                    Thread.Sleep(500);
                    arduino.Close();
                }
                catch
                {
                    Thread.Sleep(1);
                }
            }
        }
    }

    // Event handler for SkeletonPointToScreen_____________________________
    private Point SkeletonPointToScreen(SkeletonPoint skelpoint)
    {
    //Convert point to depth space
    DepthImagePoint depthPoint  =
this.sensor.CoordinateMapper.MapSkeletonPointToDepthPoint(skelpoint,
DepthImageFormat.Resolution640x480Fps30);
return new Point(depthPoint.X, depthPoint.Y);
}
    // Event handler for DrawBone_____________________________________________
    private void DrawBone(Skeleton skeleton, DrawingContext
                          drawingContext, JointType jointType0,
                          JointType jointType1)
    {
        Joint joint0 = skeleton.Joints[jointType0];
        Joint joint1 = skeleton.Joints[jointType1];
```

```
            //If we can't find either of these joints then exit
            if(joint0.TrackingState == JointTrackingState.NotTracked ||
               joint1.TrackingState == JointTrackingState.NotTracked)
            {
                return;
            }
            //Don't draw if both points are inferred
            if(joint0.TrackingState == JointTrackingState.Inferred ||
               joint1.TrackingState == JointTrackingState.Inferred)
            {
                return;
            }
            Pen drawPen = this.inferredBonePen;
            if(joint0.TrackingState == JointTrackingState.Tracked ||
               joint1.TrackingState == JointTrackingState.Tracked)
            {
                drawPen = this.trackedBonePen;
            }
            drawingContext.DrawLine(drawPen, this.SkeletonPointToScreen
            (joint0.Position),this.SkeletonPointToScreen(joint1.Position));
        }

        // Event handler for starting button in application window _______
        private void pokreni_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                this.sensor.Start();
            }
            catch (IOException)
            {
                this.sensor = null;
            }
        }

        // Event handler for stoping button in application window
        private void zaustavi_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                this.sensor.Stop();
            }
            catch (IOException)
            {
                this.sensor = null;
            }
        }
    }
}
```

11.1.2. Arduino aplikacija (Arduino IDE)

```
#include <ShiftStepper.h>

#define DATPIN  11
#define SCLPIN  13
#define LATPIN   7
#define MRPIN    8
#define INDPIN   2

byte intByte_0;
byte intByte_1;
byte intByte_2;
byte intByte_3;
byte intByte_4;
byte intByte_5;
byte intByte_6;

// Sekvenca aktiviranja faza {0001,0010,0100,1000}
static const uint8_t stepSequence[4] = {0x1, 0x2, 0x4, 0x8};

// Postavljanje kombinacije priključaka motora na shield
// Oznake priključaka na shieldu na koje je spojen prvi motor
static const uint8_t motChans0[__channelsPerMotor__] = {0,1,2,3};

// oznake priključaka na shieldu na koje je spojen drugi motor
static const uint8_t motChans1[__channelsPerMotor__] = {4,5,6,7};

shiftChain *myChain = 0;
shiftStepMotor motor0(__channelsPerMotor__, stepSequence, motChans0);
shiftStepMotor motor1(__channelsPerMotor__, stepSequence, motChans1);

shiftDevice *motors[2] = {&motor0, &motor1};
shiftSixteen board0(2, motors);
shiftBoard *boards[1] = {&board0};
shiftChain storeChain(1, boards, DATPIN, SCLPIN, LATPIN, MRPIN, INDPIN);



// Timer za odbrojavanje vremena između koraka
ISR (TIMER2_0VF_vect)
{
      myChain→doTick();
}

void setup() {
      myChain = &storeChain;
      Serial.begin(9600);
      Serial.println("I live!");
      myChain→startTimer(__preScaler32__, 0, 2);
```

```
	Serial.println("Timer started");
}

// Glavna petlja programa
void loop() {
	shiftStepMotor *thisMotor;
	shiftBoard *thisBoard;

	if (Serial.available() == 7)
	{
		//Čitanje buffera podataka
		inputByte_0 = Serial.read();
		delay(50);
		inputByte_1 = Serial.read();
		delay(50);
		inputByte_2 = Serial.read();
		delay(50);
		inputByte_3 = Serial.read();
		delay(50);
		inputByte_4 = Serial.read();
		delay(50);
		inputByte_5 = Serial.read();
		delay(50);
		inputByte_6 = Serial.read();
		delay(50);
	}
	// Poruka s podacima od Kinecta – ako je prva vrijednost 16
	if(inputByte_0 == 16)
	{
		switch (inputByte_1)
		{
			case 0:                    // slucaj mot_x = 0
			// Prva os se ne pomiče
			((shiftStepMotor*)myChain→getBoard(0)→getDev(0))→doStep
			s(0, 0);
			break;
			//-------------------------------------------------------
			case 1:                    // slucaj mot_x = 1
			val_x = inputByte_2;       // spremanje targeta za pomak
			//_____________________________________
		switch (inputByte_3)
		{
			case 0:                 // slucaj jednog smjera dir_x = 0
			((shiftStepMotor*)myChain→getBoard(0)→getDev(0))→doStep
			s(val_x, 255);
			break;
			//----------------------------------
			case 1:                 // slucaj drugog smjera dir_x = 1
			((shiftStepMotor*)myChain→getBoard(0)→getDev(0))→doStep
			s(-val_x, 255);
```

```
//_____________________________ podaci za drugi motor
      switch (inputByte_4)
      {
            case 0:                    // slučaj mot_y = 0
            // Druga os se ne pomiče
            ((shiftStepMotor*)myChain→getBoard(0)→getDev(1))→doStep
            s(0, 0);
            break;
            //-------------------------------------------------------
            case 1:                    // slucaj mot_y = 1
            val_y = inputByte_5;       // spremanje targeta za pomak
            //_______________________________

      switch (inputByte_6)
      {
            case 0:                 // slucaj jednog smjera dir_y = 0
            ((shiftStepMotor*)myChain→getBoard(0)→getDev(1))→doStep
            s(val_y, 255);
            break;
            //-----------------------------------
            case 1:                 // slucaj drugog smjera dir_y = 1
            ((shiftStepMotor*)myChain→getBoard(0)→getDev(1))→doStep
            s(-val_y, 255);
            break;
            }
      break;
}
//_____________________________________ kraj podataka za drugi motor
break;
}
break;
}

// Čišćenje memorijskih lokacija
inputByte_0 = 0;
inputByte_1 = 0;
inputByte_2 = 0;
inputByte_3 = 0;
inputByte_4 = 0;
inputByte_5 = 0;
inputByte_6 = 0;
}

// Poruka s podacima od robotske ruke – ako je prva vrijednost 14
if(inputByte_0 == 14)
{
      switch(inputByte_1)
      {
            case '0':
            ((shiftStepMotor*)myChain→getBoard(0)→getDev(0))→doStep
            s(0, 0);
```

```
                    break;
                    case '1':
                    prva_os_val = inputByte_2;
                    ((shiftStepMotor*)myChain→getBoard(0)→getDev(0))→doStep
                    s(prva_os_val, 255);

              switch(inputByte_3)
              {
                    case '0':
                    ((shiftStepMotor*)myChain→getBoard(0)→getDev(1))→doStep
                    s(0, 0);
                    break;
                    case '1':
                    druga_os_val = inputByte_4;
                    ((shiftStepMotor*)myChain→getBoard(0)→getDev(1))→doStep
                    s(druga_os_val, 255);
                    break;
              }
        }

        //Čišćenje memorijskih lokacija
        inputByte_0 = 0;
        inputByte_1 = 0;
        inputByte_2 = 0;
        inputByte_3 = 0;
        inputByte_4 = 0;
        inputByte_5 = 0;
        inputByte_6 = 0;
        }
}
```

11.1.3. UR5 programski kod (URscript)

Robot Program

- MoveJ
  - Waypoint_1
- socket_open("192.168.123.200", 200)
- tcp1:= get_target_tcp_pose()
- socket_send_string(tcp1)
- Wait: 1.0
- MoveJ
  - Waypoint_2
- tcp2:=get_target_tcp_pose()
- socket_send_string(tcp2)
- Wait: 1.0
- MoveJ
  - Waypoint_3
- tcp3:=get_target_tcp_pose()
- socket_send_string(tcp3)
- Wait: 1.0
- MoveJ
  - Waypoint_4
- tcp4:=get_target_tcp_pose()
- socket_send_string(tcp4)
- Wait: 1.0

11.2. CD-R disk